ANALYSIS OF ERROR CONTROL AND CONGESTION CONTROL PROTOCOLS

by

Amarnath Mukherjee

Computer Sciences Technical Report #960
August 1990

ANALYSIS OF ERROR CONTROL AND CONGESTION CONTROL PROTOCOLS

by

AMARNATH MUKHERJEE

A thesis submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
Computer Sciences

at the
UNIVERSITY OF WISCONSIN — MADISON
1990

Abstract

This thesis presents an analysis of a class of error control and congestion control protocols used in computer networks.

We address two kinds of packet errors: (a) independent errors and (b) congestion-dependent errors. Our performance measure is the expected time and the standard deviation of the time to transmit a large message, consisting of N packets.

The analysis of error control protocols assuming independent packet errors gives an insight on how the error control protocols should really work if buffer overflows are minimal. Some pertinent results on the performance of go-back-n, selective repeat, blast with full retransmission on error (BFRE) and a variant of BFRE, the Optimal BFRE that we propose, are obtained.

We then analyze error control protocols in the presence of congestion-dependent errors. We study the selective repeat and go-back-n protocols and find that irrespective of retransmission strategy, the expected time as well as the standard deviation of the time to transmit N packets increases sharply the face of heavy congestion. However, if the congestion level is low, the two retransmission strategies perform similarly. We conclude that congestion control is a far more important issue when errors are caused by congestion.

We next study the performance of a queue with dynamically changing input rates that are based on implicit or explicit feedback. This is motivated by recent proposals for adaptive congestion control algorithms where the sender's window size is adjusted based on perceived congestion level of a bottleneck node. We develop a Fokker-Planck approximation for a simplified system; yet it is powerful enough to answer the important questions regarding stability, convergence (or oscillations), fairness and the significant effect that delayed feedback plays on performance. Specifically, we find that, in the absence of feedback delay, a linear increase/exponential decrease rate control algorithm is *provably* stable and fair. Delayed feedback, however, introduces cyclic behavior. This last result not only concurs with some recent simulation studies, it also expounds quantitatively on the real causes behind them.

Acknowledgments

I most sincerely acknowledge:

a) Professors Larry Landweber and John Strikwerda for teaching me all the good stuff, caring about me and my work and for all the patience when I goofed,

b) Professor Mary Vernon, for her quality help all along, including, working on my slides into the wee-hours of the morning,

c) Professors Bart Miller and Tom Leonard, for most graciously agreeing to be in my committee,

d) Professor Miron Livny, for providing me his DeNet simulation tool,

e) Sheryl Pomraning and Laura Cuccia, for being ever so helpful and nice,

f) Cheng Song and Mitch Tasman, for being wonderful officemates,

g) my friends, in reverse alphabetic order: Vikram/Sarita, Tsuei, Sriram Vajapeyam, Shahram, Scott/Lisa, Rustam, Prasad, Parikshit, Osty, Narendran, George Bier, Gautam/Moneka, Deshpande, Chinmoy/ Sumita Roy and family, Basu, Atul Parekh, Arun/Purabi Datta and family, Argha, Ananda/Mousumi,

h) my chess buddies, Bill Olk, Bill Arvola, Dean Olson, Guy Hoffman, Joseph Albert, Karl Zanguerle, Larry Butler, Marc Ingeneso and Senthil Kumar,

i) IBM, for providing me financial support during 1989-90,

j) the Computer Sciences Department, the Memorial Union Terrace, and Madison city in general, for making these years most wonderful.

I am also deeply indebted to my parents and my family for their love, understanding and support.

CONTENTS

Chapter 1
Introduction

1.1. Problem Statement and Motivation

This thesis presents an analysis of a class of protocols used in computer networks. The analysis of these protocols is important because

a) it gives an estimate of the performance of these protocols that is otherwise hard to obtain,
b) it quantifies the relative importance of different performance issues and
c) it identifies quantitatively the cause of any undesirable behavior.

As an example, consider the computer network[1] shown in Figure 1.1. The network consists of *nodes* which are interconnected with *channels*. Each node serves as a switching element that routes packets from one of several inputs to one of several outputs. It has limited buffering capabilities to deal with sudden bursts in traffic. Users, located outside of this network in *hosts* communicate with each other through this network. They do so by means of predefined *protocols,* which specify the rules of interaction between two semi-autonomous units. There is an entire gamut of protocols that are defined for computer communication. These provide different services like reliable data delivery, directory service, multicasting, etc. The quality of service that the protocol provides may vary depending upon the perceived importance of that service. For instance, a protocol could provide reliable and sequential delivery of packets as in X.25, or it could make only a best effort at delivering individual packets as in IP. In the latter case, the sender and the receiver may agree on a protocol for error recovery at a higher level. Protocols may

[1] or rather its queueing network model

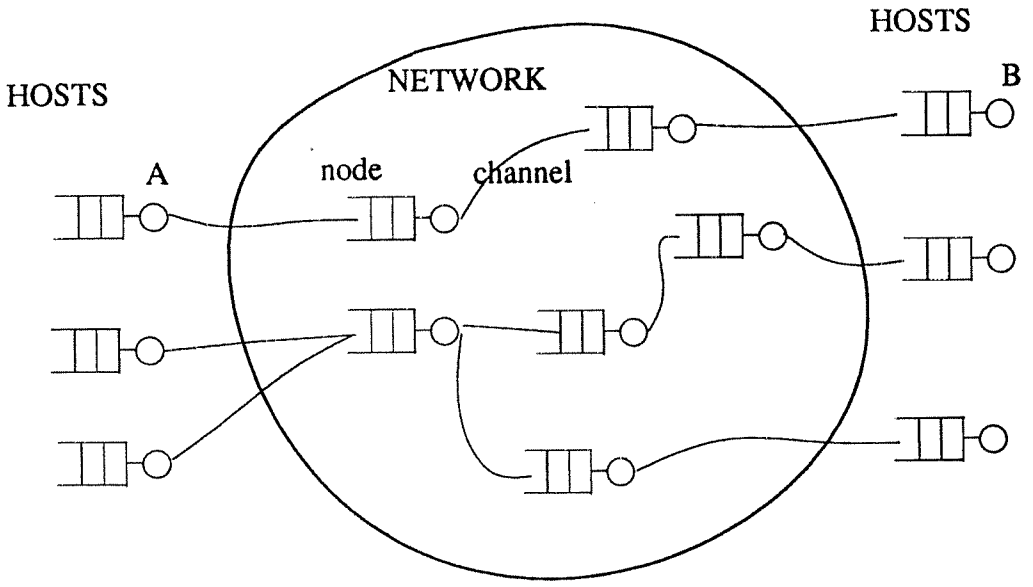

Figure 1.1. An example computer network.

also specify a fixed or variable *rate* of transmission or the maximum number of packets (the *window* size) that the sender could have outstanding at any time before receiving an acknowledgment from the receiver. These protocols are called *flow control* and *congestion control protocols.* Briefly, flow control attempts to alleviate mismatch in speeds between the end-points while congestion control protects the network elements from being overrun by fast transmitters. It is often easy to devise a protocol but *difficult to estimate or verify how it will perform.* A further complication arises from the fact that a protocol may also have *side effects* on the performance of other protocols. Thus a poor congestion control protocol could, for instance, drive up the error rates artificially to the point where the chosen error recovery protocol is sub-optimal. It is therefore important to develop methods for assessing not only the performance of these protocols in isolation, but also to consider their interactions if necessary, using either analytical techniques or simulations and experiments.

Protocols create interesting and intriguing phenomena which can be expressed mathematically and analyzed for their performance. In this thesis, we apply mathematical analysis to the specific problems of understanding

error control and congestion control protocols.

Error control protocols, as the name implies, are used to recover from errors. When some user, say at host A in Figure 1.1, submits a *message* to the network to be transmitted to another user, say at host B, the message is usually split into *packets* which are transmitted over the network and reassembled at the other end. A packet in transit encounters one or more channels, (e.g., satellite, copper wire or optical fiber), and nodes (or *routers*), which route the packet to the destination. These intermediate elements can induce *errors* in the packet in that either the packet could get *garbled,* or *dropped* altogether. The former is usually due to random electrical noise in the channels while the latter is due to buffer overruns at the nodes and is caused by contention for resources, a phenomenon often referred to as *congestion.*

Protocols that are implemented to recover from packet errors are called *error control protocols;* those that attempt to alleviate congestion are called *congestion control protocols.* With respect to performance, their interaction is closely related. The overall *end-to-end* performance for a user depends on how well a combination of the two protocols performs. The use of fiber optic technology has significantly decreased network errors in channels; hence the load on the error control protocol depends heavily on the success (or failure) of the congestion control protocol because the latter affects congestion-related losses. Conversely, an error control protocol could also aggravate congestion in the network, for example, by introducing a large number of retransmitted packets. *An analysis of end-to-end user performance must therefore study these two protocols in unison rather than in isolation.* Previous work has, however, not addressed these two issues simultaneously. In our study, we explicitly address errors that are caused by congestion.

A related and perhaps more important problem in congestion control is the transient analysis of dynamic congestion control protocols [RaJa 88, Jac 88]. These protocols adjust the sender's window size based on perceived congestion level of a bottleneck node. To analyze their performance, one needs to study the stochastic behavior of a queue with dynamically changing

input rates which are based on feedback. The issues that need investigation are

a) how quickly does the system adapt to changing environments?

b) does it stabilize or show cyclic behavior?

c) is the protocol fair?

d) how do the system parameters (like delay, multiple hops, other competing users, etc.) change any of the above?

Precise answers to these questions that either support or point to flaws in common intuition are certainly worthwhile, and are the subject of our study.

1.2. Overall Approach and Summary of Results

Our study focuses on the statistics of the *time to complete a multi-packet end-to-end message transfer.* The measures used in previous analyses on error control protocols were *maximum channel throughput* or *queue length characteristics at the sender,* given assumptions of packet arrival rates and distributions [AnPr 86, BrMo 86, ToWo 79, MoQiRa 87]. For a user who is interested in accessing files, or in remote procedure calls over a network, however, *end-to-end performance* is a more relevant measure. Hence, we choose the time to successfully transmit a message of N packets as our *performance measure.* The only other study that incorporates this performance measure is one by Zwaenepoel [Zwa 85], who analyzed the *stop-and-wait* protocol and *blast protocol with full-retransmission-on-error (BFRE)* for a multi-packet message assuming independent packet errors.

Our first contribution is a theoretical analysis of the *go-back-n* and *selective repeat* protocols under the same assumptions as Zwaenepoel's and a comparative study of these and BFRE in a local area network environment. We derive expressions for the expectation, variance and the distribution of

time to transmit N-packets using the go-back-n and selective repeat protocols. These are compared to the expressions for BFRE. We conclude that go-back-n performs almost as well as selective repeat while BFRE is stable only for a limited range of message sizes and error rates. Since go-back-n has a simpler state machine than selective repeat, it is therefore the protocol of choice. We also present a variant of BFRE, the *optimal BFRE,* which *optimally* checkpoints the transmission of a large message. This is shown to overcome the instability of ordinary BFRE. Moreover, its simple state machine seems to take full advantage of the low error rates of local area networks. We further investigate go-back-n by generalizing the analysis to an upper layer transport protocol, which is likely to encounter among other things, *variable* delays due to protocol overhead, multiple connections, process switches and operating system scheduling priorities.

Our next contribution is the analysis of error control protocols when errors are *congestion-dependent.* Most earlier work assumed statistically *independent* packet errors. This is not a very realistic assumption in today's networks because buffer overruns are the principal source of errors and these errors are correlated. In fact, it is more likely for an error to occur when one has already occurred than when none has. We develop models of congestion which help evaluate the go-back-n and the selective repeat protocols. The congestion model is based on the *empirical evidence*[2] that in window based flow control protocols, a connection's loss rates increase monotonically with the number of packets that it has outstanding in the network.

A third contribution of this research is the theoretical analysis of a class of *congestion control protocols* that rely on *feedback.* These protocols are *adaptive* in that they require the end-points to *adjust* the *window* size or the *rate* of transmission when congestion sets in at some intermediate node. We develop from first principles, a Fokker-Planck equation for the evolution of the joint probability density function of queue length and arrival rate at this node. This approximates the *transient* behavior of a queue subjected

---

[2] See Figure 7 in [SSSGJ 88]. This particular *observation* was, however, *not* made by the authors. Also see the note to Figure 9 in [Jac 88] for further evidence.

to an adaptive rate-control algorithm. It can answer important questions regarding *stability* (or oscillations) and *fairness* of a particular adaptive algorithm as well as the significant effect of *delayed feedback* on the conclusions. For instance, in the absence of feedback delay, senders using the Jacobson-Ramakrishnan-Jain (JRJ) algorithm [Jac 88, RaJa 88,90], (or rather, an equivalent rate-based algorithm) can be shown to *converge* to an equilibrium. Further, this algorithm is *fair* in that all the sources sharing this resource get an equal share if they use the same parameters for adjusting their rates. The exact share of the resource when the different sources use different parameters can also be determined from this analysis.

A delay in the feedback information will cause the system to exhibit oscillatory behavior. These oscillations converge to a limit cycle. If different sources get the feedback information after *different* amounts of delay, then the algorithm can also be *unfair,* i.e., they do not get equal throughput. In a simulation study of the same protocol, Zhang observed oscillations in the queue length at intermediate nodes [Zha 89]. She also observed that connections with larger number of hops received a poorer share of a shared resource than those with a smaller number of hops. Our analysis not only concurs with her simulations, it also *explains* the *reasons* for the behavior of the protocols she simulated. The oscillations are due to delay in feedback; the unfairness is *partly* due to the larger (feedback) delay suffered by the longer connections as compared to the shorter ones.

1.3. Thesis Outline

Chapter 2 surveys related work in error control and congestion control protocols. It also has all the relevant definitions. In Chapters 3-5, we study error control protocols. First, we reduce the degrees of freedom to the case when errors are statistically independent, the network consists of a single hop and there are no windowing effects. This study is presented in Chapter 3. We investigate the performance of the *go-back-n* protocol, the *selective repeat*

protocol, the *blast protocol with full retransmission on error (BFRE)* and a variant of BFRE which we call the *optimal blast protocol.* We find that the BFRE protocol becomes unstable much faster with respect to message size than the go-back-n protocol or the selective-repeat protocol. However, since BFRE has a very simple state machine, it makes other design issues much simpler and efficient (for example, the network interface design of Kanakia and Cheriton [KaCh 88]). It also seems ideally suited for an environment where host processing time is a significant amount of the total time, precisely because the amount of 'work' to be done by the host is reduced. This is the motivation for our *optimal blast protocol* which performs well for both large and small message sizes.

In Chapter 4, the assumption of infinite windows is removed. The single hop network is also generalized to any arbitrary network. Packet errors are still assumed to be independent of each other. We find that the window closing effect has a minimal effect on the *analysis* of go-back-n. The window-effects and the error-effects are *quasi-independent* in that they could be studied separately and the results put back together in an obvious way. Unfortunately, no such relationship was found to hold for selective repeat.

In Chapter 5, the assumption of independence of packet errors is removed. The errors are *congestion-dependent.* We first develop a new *congestion model,* which gives the probability of error as a function of the number of packets that are outstanding in the network. The congestion model is incorporated into the *protocol models* of go-back-n and selective repeat to yield two separate continuous time Markov processes. Each Markov process has an initial state corresponding to the beginning of a message transmission and a final state corresponding to its end. A transient solution of the Markov process yields the expected time to transmit an N-packet message and its variance. We find that irrespective of retransmission strategy used, the expected time as well as the standard deviation of the time to transmit N packets increases sharply if the window size is large in the face of heavy congestion. However, if the congestion level is low, the two retransmission strategies perform similarly.

In Chapter 6, we develop a *theory for dynamic congestion control algorithms.* The algorithm of Jacobson-Ramakrishnan-Jain [Jac 88, RaJa 88,90], (the 'JRJ'- algorithm) is a special case of this general framework. In the JRJ algorithm, when congestion is detected (by implicit or explicit feedback), the window size is *decreased multiplicatively.* However, when there is no congestion, it is *increased linearly* — to probe for more bandwidth. While this seems to be a good adaptive algorithm, it is far from clear as to what *values* the parameters for increasing or decreasing the window size should take. Further, it is not *provably* clear if the algorithm is *fair* or *stable* and if so, *under what circumstances.*

To understand the behavior of dynamic congestion control algorithms, we study the behavior of a queueing system with a time varying input *rate.* This rate is adjusted periodically based on some feedback that the transport endpoint receives about the state of the queue. Let $g(\lambda, q) = d\lambda(t,q)/dt$ be the *rate control algorithm,* where $\lambda(t)$ is the input rate at time, t. As an example, $g(\cdot)$ could be the following function:

$$g(\lambda, q) = \frac{d\lambda}{dt} = \begin{cases} +C_0, & \text{if } q \leq \bar{q}, \\ -C_1\lambda, & \text{if } q > \bar{q}. \end{cases} \tag{1.1}$$

where $\bar{q}$ is some threshold queue length. This is the rate-equivalent of the window based JRJ-algorithm (note the linear increase and exponential decay components).

A transient analysis of this queueing system is difficult. We have approximated its behavior by a 2-dimensional Fokker-Planck equation. The result is a second-order partial differential equation for the joint probability density function $f(\cdot)$ of the queue length and the arrival rate:

$$\frac{\partial f}{\partial t} + \nu\frac{\partial f}{\partial q} + \frac{\partial gf}{\partial \nu} = \frac{1}{2}\sigma^2\frac{\partial^2 f}{\partial q^2} \tag{1.2}$$

where $f(t, q, \nu)$ is the joint probability distribution of q and ν, $\nu(t) = \lambda(t) - \mu$ is the instantaneous mean queue growth rate, μ is the instantaneous mean service rate of the queue and σ^2 is the variance of queue growth rate. Equation 1.2 is studied in detail in Chapter 6. There we find that the linear

increase and exponential decrease algorithm given by Equation 1.1 is inherently *stable* if there is no delay in feedback, i.e., it converges to the correct value of $\lambda = \mu$ and threshold queue-length, $\bar{q}$. The effect of the parameter values C_0 and C_1 are also studied.

Introduction of feedback delay however adds *oscillations* which settle down to a *limit cycle,* i.e., a cyclic pattern that is constant in the limit. This cyclic pattern concurs with simulation results by Zhang [Zha 89]. The proof of the existence of a limit cycle, we believe, is a new result. The diameter of the limit cycle (or equivalently the magnitude of the oscillations) is sensitive to the parameters C_0, C_1 and the feedback delay. For instance, for a fixed C_0 and feedback delay, a larger C_1 *increases* this diameter. So, while in the absence of feedback delay, a larger C_1 boosts the speed of convergence, in the presence of delay, it causes wilder oscillations. The size of the oscillations also increase with C_0 and feedback delay.

Chapter 2

Related Work

2.1. Outline

This chapter surveys the literature in error control and congestion control protocols. Since these two issues have been studied independently of each other in the past, we split this chapter into two major sub-sections. First, we review related work on error control protocols and then on congestion control protocols.

2.2. Error Control Protocols

2.2.1. Background

Two error control protocols that we are primarily interested in investigating are the *go-back-n* and the *selective repeat* protocols. In addition, we shall also consider the *Blast protocol with full retransmission on error (BFRE)* and a variation of this protocol the *optimal BFRE,* which we propose in Chapter 3. We shall however not consider the *stop-and-wait* protocol because it is known to perform poorly [Zwa 85].

In the next few sub-sections, we shall survey previous work on the go-back-n and the selective repeat protocols. Numerous variants of these two protocols have been proposed in the literature [Sha 75, Mor 78, LiYu 78, Tow

79]. The design of these protocols seems to be an easy task, whereas their analysis and performance evaluation proves to be very difficult. Nevertheless, some of them have been analyzed to determine either their *queue length* statistics or the *maximum throughput* that they can deliver.

The analyses of these protocols have usually assumed packet errors to be independent of each other [ToWo 79, Tow 79, Kon 80, AnPr 86, BrMo 86]. In addition, the roundtrip delay is assumed to be fixed (deterministic) and the window is assumed to be open at all times.[1]

Fujiwara *et al,* [Fu 78], assumed a burst error model, first suggested by Gilbert [Gil 60],[2] to analyze the throughput of go-back-n in conjunction with forward error correction. However, they show numerical results for independent errors only and mention that the burst model behaves similarly.

The outline of the rest of Section 2.2 is as follows. We first review preliminary definitions of the basic go-back-n and the selective repeat protocols. As mentioned earlier, numerous small variations of these protocols have been proposed. We discuss interesting results from the literature on these protocol variants. The studies that involve queueing analysis are presented first; those involving throughput analysis are presented next.

2.2.2. Basic Protocol Definitions

Assuming a sliding window flow control, the basic go-back-n and selective repeat protocols work as follows: When a packet is successfully received at the receiver, it is always acknowledged (or ACKed) if it is "in-sequence." In the case of selective-repeat, the receiver may also ACK out of sequence

---

[1] These assumptions do not reflect the properties of real networks. It has been demonstrated that the roundtrip delay may fluctuate considerably [Jac 88] and the window can therefore close too.

[2] The Gilbert error model is a correlation model for errors in satellite channels based on a two state Markov process. In one state, the probability of error is zero; in the other, it is equal to p. The transition probabilities between these states completely specifies the error model.

data, but will not deliver them to its 'user' at the receive-end. In both cases, an error is detected at the sender by either a timer interrupt or a NACK from the receiver. At this point, if the sender backs up to the first packet in error, i.e., the first packet that is not yet ACKed, and restarts the transmission, the strategy is referred to as go-back-n [Tan 81]. If, on the other hand, the sender retransmits only those packets which are in error, the strategy is called selective-repeat. In go-back-n, buffering and reassembling of a message at the receiver is much simpler than in selective-repeat, but at the potential cost of retransmitting many more packets. Selective repeat on the other hand, may require *large receive buffers* if the propagation delay and window size are large. The go-back-n protocol is not *required* to have more than one receive buffer, although it may buffer packets waiting to be sent to the user.

Almost all previous work has attributed the 'n' in go-back-'n' to be the number of packets that the sender backs up by (and retransmits) in case of an error. 'n' is assumed to be a constant in these studies. This makes sense if the sender is transmitting a *full* window of packets *all the time* and the window size is 'n'. However, since that is not the case in real networks, we have chosen to ignore this interpretation of 'n'. Instead, we explicitly use the window size wherever necessary, thereby permitting more realistic scenarios with variable number of packets in the pipe.

2.2.3. Queueing Analysis Results

In this sub-section, we outline the queueing analyses for go-back-n, the 'stutter' go-back-n [Tow 79], and the selective repeat protocols. The go-back-n analysis is due to Towsley and Wolf [ToWo 79]. The 'stutter go-back-n' protocol is a modified go-back-n protocol. It was proposed and analyzed by Towsley [Tow 79]. The selective repeat results are due to Konheim [Kon 80] and Anagnastou and Protonataraious [AnPr 86].

2.2.3.1. Go-back-n

Assumptions

(1) Time is divided into subintervals of duration Δ, called slots. All results are normalized with respect to Δ, i.e., $\Delta = 1$.

(2) Packets arrive at the sending multiplexor just prior to the beginning of each slot. The number of packets which arrive in any slot is given by the random variable D. These are independent and identically distributed (i.i.d) with the distribution $p_k = P[D = k], k = 0, 1, 2, \cdots$. The distribution has mean $\mu_D = E[D]$, and variance ${\sigma_D}^2 = E[(D - \mu_D)^2]$.

(3) The packets are served on a first-come-first-served basis.

(4) The number of packets queued in the multiplexor at the beginning of the jth slot is given by L_j.

(5) The process has been running long enough so that the statistics of L_{j-1} and L_j are identical. These are therefore replaced by the generic random variable L.

(6) The queue has unlimited capacity.

(7) The roundtrip delay is a fixed number of slots, s.

Analysis

Because of errors in the channel, a packet may be transmitted more than once. Let N_i be the total number of times that packet i is transmitted (including retransmissions). Assuming N_i are i.i.d. random variables, represent them by N. Let N have a mean μ_N and variance ${\sigma_N}^2$. To aid the analysis, a packet is converted into a, so called, *'slacket'*, on arrival. A slacket is a fictitious quantity which represents the number of slots that will be necessary to transmit the packet. Let the slacket size be denoted by the random variable

M with mean μ_M and variance ${\sigma_M}^2$. M and N are related by the equation $M = 1 + (N-1)(1+s) = N(1+s) - s$, since the first transmission takes one slot, but all subsequent retransmissions take $(1+s)$ slots each. Thus

$$\mu_M = (1+s)\mu_N - s,$$

and

$${\sigma_M}^2 = (1+s)^2{\sigma_N}^2$$

The mean queue length at the sender for the go-back-n protocol is then given by (see [ToWo 79] for details):

$$\mu_L = \frac{\mu_D(\mu_M + 2s)}{2} + \frac{{\mu_D}^2{\sigma_M}^2 + \mu_M{\sigma_D}^2}{2(1-\mu_D\mu_M)}$$

Results

Towsley and Wolf plot solutions for the expected queue length assuming a Poisson arrival process and a geometric error probability distribution. Perhaps the most interesting result is the effect of roundtrip delay (s), and the error probability (p). The queues grow exponentially as p increases. The performance also gets worse with increasing s, but the effect is much slower. For actual quantitative results, the reader is referred to the original paper [ToWo 79].

2.2.3.2. Stutter go-back-n

The protocol

The performance of the go-back-n protocol degrades with higher error rates and higher roundtrip delays. To improve the performance of go-back-n, Towsley [Tow 79] proposes the 'stutter' go-back-n protocol, which is the original go-back-n with the following modification: during periods when the channel would normally be idle under go-back-n, the sender repeatedly transmits the last unacknowledged packet, if any, residing in the queue. Towsley derives the queue length statistics for this protocol, along with that of an 'idealized' retransmission protocol, which in some sense represents the upper bound on the performance.

The stutter go-back-n protocol has some complications that need clarification. A packet, say i, that has been repeatedly transmitted when the channel was idle may have several ACKs/NACKs return in consecutive slots. Assume that there is at least one other packet behind it in the queue now. Packet i, which is at the head of the queue, should only be retransmitted if all the acknowledgment packets are NACKs. The sender therefore needs to keep track of the number of repeated transmissions of a packet and the number of acknowledgments (ACKs and NACKs) that have returned. Notice that if we allow acknowledgment losses, this tracking method fails. The stutter go-back-n protocol is also not very effective in environments where errors are congestion-related; unnecessary multiple transmissions of the same packet in a congested network is highly undesirable.

Assumptions

The assumptions for the analysis of this protocol are the same as that for the go-back-n protocol discussed earlier.

Analysis

The analysis of this protocol is cumbersome and the formulae give little intuitive insight. The methodology however, is similar to the go-back-n analysis of Towsley and Wolf [ToWo 79]. We therefore refer the reader to [Tow 79] for the detailed analysis.

Results

At low utilizations and/or low error rates, stutter go-back-n cannot improve much over go-back-n because there is not much queueing at these loads. For very high utilizations, stutter go-back-n cannot improve much over go-back-n either, because idle channel bandwidth is hard to come by. However, for moderately utilized systems and high error rates, stutter go-back-n improves considerably over go-back-n. For instance, if $s = 10, p = 0.1, \rho = 0.6$, (where ρ is the utilization), the average delay as compared to normal go-back-n is reduced by 20%. If $s = 20$, the average delay is reduced by 30%. For $s = 10, p = 0.5, \rho = 0.6$, the difference is more than 50%. Notice the large values of s and p in these examples. It is only for such parameter values that go-back-n performs poorly.

2.2.3.3. Selective repeat protocol

The results here are due to Anagnostou and Protonotarious [AnPr 86] and Konheim [Kon 80].

Assumptions

(1) The system is a slotted multiplexor as in the earlier analyses.
(2) The packet arrival process is the same as that in [ToWo 79] and [Tow 79].
(3) Transmission errors are independent of each other.
(4) The queue at the sender has unlimited capacity.
(5) The roundtrip delay is a constant, denoted by s. Also, if a packet is transmitted at time m, then either an ACK or a NACK is received in the slot $(m+s-1, m+s)$.
(6) At the beginning of a slot, the first packet in the queue is transmitted, unless a NACK arrives for an earlier packet in the previous slot. In the latter case, the packet in error is transmitted. This is the property of the selective repeat protocol.

Analysis

The analysis of the selective repeat protocol turns out to be considerably more complex than that of go-back-n. The resulting solutions are algorithmic in nature. We briefly outline the method of analysis.

The basic idea is to use a discrete time Markov process which describes the state of the system at any time t (t is an integer). The state of the system that is adopted by both [Kon 80] and [AnPr 86] is

$$S(t) = (Q(t); r_1(t), r_2(t), \cdots, r_s(t))$$

where $Q(t)$ = queue length at $t+1$, and

$$r_i(t) = \begin{cases} 1, & \text{if a transmission was attempted at } t-i+1, \\ 0, & \text{otherwise} \end{cases}$$

The next step is to determine the state transition matrix for the process. One has to account for the arrivals in the current slot. This will affect $Q(t)$.

Depending upon whether or not a transmission has taken place s time units earlier (i.e., if $r_s(t) = 1$ or 0, respectively), there may be an ACK/NACK returning, or nothing at all. Also, since only one packet is transmitted in a slot, at most one ACK/NACK may return in a slot. This, coupled with the probability of error gives another set of transitions. If no NACK arrives and $Q(t) > 0$, then a new transmission is attempted. Note that r_i shifts once to the right and r_1 is determined by whether or not a transmission is attempted in the current slot. The state transition matrix is thus completely specified.

Assuming that a steady state is finally reached, the Markov chain described above is analyzed (algorithmically) in a standard way to determine the steady state probabilities. Summing over all possible vectors $(r_1, \cdots, r_s)$, and a value of queue length, say q, gives the steady state probability distribution of the queue length, $P[Q = q]$. The mean queue length is then easily obtained.

Results

The principal results that are presented are curves for the mean queue length versus packet error rate for different interarrival times. The interarrival times are assumed to be geometrically distributed. As expected, the curves show poorer performance for higher arrival rates and higher error probabilities.

2.2.4. Maximum Channel Throughput Results

This set of studies deals with determining the maximum channel throughput that is obtainable from a given error control protocol. Various subtle variations of the go-back-n protocol have been proposed, see for example [BrMo 86, Bir 81, LiYu 80, Mor 78, Sha 75]. The proposal by Bruneel

and Moeneclaey [BrMo 86] is the most general protocol. The authors of that paper also argue that it is the best. We concur with that view for the case when errors are independent. (For congestion dependent errors, this protocol will need re-evaluation). We discuss the results of this paper in detail. The other protocols are inferior and we only compare them briefly with the Bruneel and Moeneclaey protocol.

2.2.4.1. Bruneel and Moeneclaey Protocol

The protocol

The major modification to the go-back-n protocol that Bruneel and Moeneclaey propose is to transmit *multiple copies* of each data packet instead of a single copy. The tradeoff here is between the cost of not transmitting a new packet in the next slot versus that of finding that an error has occurred after a roundtrip delay and retransmitting all over again. For high network error rates it may be worthwhile to send multiple copies of the data so that at least one of them reaches correctly. The performance improvement may be significant for large roundtrip delays. Bruneel and Moeneclaey derive the optimal number of packets that should be transmitted in each 'cycle' to maximize throughput. This value is of course dependent on the packet error probability and the roundtrip delay.

Assumptions

(1) Packet errors are independent of each other; let p be the packet error probability.
(2) All ACK/NACK messages are received error free at the transmitter.

(3) roundtrip delay is fixed and is equal to s.

(4) All transmissions of a packet, say i, that were undone by an error in an earlier packet are ignored for analysis purposes. Thus packet i is considered to be transmitted for the *first* time if the previous transmission of packet $i-1$ is successful. At this time, the protocol requires that m_0 copies of packet i be transmitted. If all the copies are in error, a retransmission cycle is triggered and now m_1 copies are to be transmitted; if that fails too, m_2 copies are to be transmitted and so on. The process is repeated until a positive acknowledgment for at least one copy of the packet is received.

Analysis and Results

(1) Let the optimum value of m_j be ${m_j}^*$. Then it is first shown that ${m_0}^* = {m_1}^* = \cdots = {m_j}^* =$, say, m^*. The intuition behind this is that, since packet errors are independent and roundtrip delay is fixed, there is no difference between any two different (re)transmission cycles.

(2) m^* is determined as follows. Consider the function $c(p,s)$ given by

$$c(p,s) = p^{-s} - 2s(1-p) - 1$$

Also, let $\hat{m}$ be such that

$$\frac{\hat{m} + sp^{\hat{m}}}{1-p^{\hat{m}}} \leq \frac{m + sp^m}{1-p^m}, \qquad m \leq s$$

i.e., $\hat{m}$ minimizes the expression on the right hand side. Then the optimal value, m^*, is given by

$$m^* = \begin{cases} \infty, & \text{if } c(p,s) < 0, \\ \text{any number} \geq s, & \text{if } c(p,s) = 0, \\ \hat{m}, & \text{if } c(p,s) > 0. \end{cases}$$

A consequence of this result is that the curve $c(p,s) = 0$ divides the (p,s) plane into two regions, one where $c(p,s) > 0$ and the optimum is $\hat{m}$,

and the other where $c(p,s) < 0$ and the optimum is $m^* = \infty$. When $m^* = \infty$, the idea is to keep transmitting the same packet until an ACK is received for it. The (p,s) diagram (not reproduced here) shows that for low error rates and low roundtrip delays, $m^* = 1$. However, as the error rate and the roundtrip delay increases, the value of m^* increases, albeit slowly. Thus, the original go-back-n is optimal only in a small region of the (p,s) plane. Fortunately, this also happens to be the region where most networks operate (see the curves in [BrMo 86]).

Let us next review (and compare) some of the other modifications that have been proposed. Shastry [Sha 75] suggests a modified go-back-n protocol which works as follows: until an error is detected, only a single copy of each packet is transmitted as in go-back-n; in case of an error however, the packet in error is transmitted repeatedly until a positive ACK is received for it. Restated in the Bruneel and Moeneclaey framework, $m_0 = 1$ and $m_1 = \infty$. Clearly this is suboptimal because Bruneel and Moeneclaey show that the optimal value must be the same across all retransmission cycles, assuming independent packet errors, of course. Network errors are however, bursty and Shastry's protocol may perform well in practical situations.

Birrel's retransmission scheme [Bir 81], is also a special case of the general proposal of Bruneel and Moeneclaey. The m_j 's are chosen equal to some common value n *less than* s. Notice that this cannot be optimal for the region $c(p,s) < 0$, where the optimal value is $m^* = \infty$.

The selective repeat protocol may outperform the optimal go-back-n strategy for high error rates and large roundtrip delays.

This concludes our discussion of the literature on error control strategies.

2.3. Congestion Control, Congestion Avoidance and Flow Control

2.3.1. Preliminaries

" *Congestion control* is concerned with allocating the resources in a network such that the network can operate at an acceptable performance level when the demand exceeds or is near the capacity of the network resources" [Jai 90]. The algorithms must address fair resource sharing, buffer overruns and large queues at *intermediate nodes* of the network. *Flow control* protocols are similar to congestion control protocols, except they deal with *end-to-end* congestion.

Congestion *avoidance* protocols are a *subset* of congestion control protocols. They attempt to *prevent* buffer overruns and large queues from building up. This usually requires *explicit* feedback from the network. The 'explicit binary feedback protocol' of Ramakrishnan and Jain [RaJa 88, 90] is an example of a congestion avoidance protocol.

Another class of congestion control protocols attempts to *react* to congestion by receiving *implicit* feedback information from the network (like increased roundtrip delays or detection of packet losses). Jacobson's algorithm [Jac 88] falls in this category.

Notice that the algorithms in both the above categories may attempt to react in the same way. The difference in classification comes from the way they obtain congestion information. Since both these schemes are based on reacting to network conditions based on feedback, they are also referred to as 'closed-loop' congestion control protocols.

This is in contrast to 'open-loop' congestion control protocols which have recently been proposed [SLCG 89, Zha 89, BCS 90, Gol 90]. These protocols do not rely upon feedback from the network and are gaining acceptance in high speed networks where the relatively large propagation delay makes

feedback information unreliable. Some recent algorithms in this class of congestion control protocols are the *virtual clock* protocol [Zha 89], the *leaky bucket* protocol [Tur 86, SLCG 89], the *generalized leaky bucket protocol* [BCS 90] and the *stop-and-go queueing* [Gol 90].

In this thesis, we address only protocols which use feedback information for congestion control or congestion avoidance. Accordingly, we review the literature on 'closed-loop' congestion control and congestion avoidance strategies.

2.3.2. What causes congestion?

The *capacity* of network resources (example, link speeds, number of buffers, processing capacity, etc.) is usually planned on the basis of estimated demand. Congestion is usually caused by a temporary surge of traffic. This could be due to many reasons. It could be the 'time-of-day' phenomenon: during the course of the day, certain times have more traffic than others. Or it could perhaps be due to bursty traffic (data usually has a very high peak to average ratio). Other reasons, like poor routing algorithms that create hot-spots are also possible, but we shall not address them here.

The important point that we want to stress is that there are *short term fluctuations* in queue lengths due to *bursty traffic* and (relatively) *long term fluctuations* due to, say, 'time-of-day'. Different techniques may have to be used for dealing with the two cases. To understand why, let us consider a high speed, wide area network: it has a large bandwidth-delay product that makes closed-loop feedback control ineffective for short term fluctuations. This is because the feedback information is too old. However, feedback can still be used to track (relatively) long-term traffic intensity.

We know from the results of single server queuing systems that for stability, the average arrival rate (λ) of customers into the system must be less than the average service rate (μ). Even in systems where λ is less than μ on the average, it could be greater than μ for a significant amount of time, as for

example, during peak hours. This results in what Newell [New 68] calls the *rush-hour-effect:* it takes a *very long time* for the queueing system to return to steady state once it hits rush hour. It is for this reason that freeways remain saturated long after the close of business. While Newell shows this for a single queue, we expect to see a similar phenomenon for a network of queues too. The point to note here is that *packet loss is not the only reason to avoid congestion.*

Jacobson [Jac 88], argues that 'stability' of a communication system is affected directly by dropped packets. He draws an analogy to thermodynamics and claims that, for stability, the protocol has to obey the *conservation of packets* principle. That is, for a connection in 'equilibrium', (i.e., transmitting a full window of data), a new packet should not be injected into the network until an old one leaves. Of course, stability also will be affected by large queueing delays because it could cause premature retransmissions.

In summary, congestion could be caused by short term or long term fluctuations in traffic. The result of congestion could be packet losses and/or large queueing delays. Even systems with a large number of buffers could see appreciable degradation in performance due to large queueing delays, not to mention the possibility of premature timeouts. Congestion can undermine the stability of the network.

2.3.3. Proposed Solutions

We next survey some of the solutions that have been proposed to avoid or alleviate the effects of congestion. The most interesting results are due to Jacobson [Jac 88] and Ramakrishnan and Jain [RaJa 88, 90]. These solutions, while different in detail, are similar in practice. We first discuss Jacobson's solution. Although his argument does not include a mathematical proof, his proposed modifications to BSD/TCP have greatly improved the performance of this protocol.

As mentioned before, Jacobson's goal is to maintain the conservation of packets. He identifies three ways for packet conservation to fail:

(i) A connection does not come to equilibrium.

(ii) A sender injects a packet before an old one has exited, or

(iii) Equilibrium cannot be reached because of resource limits along its path.

We summarize Jacobson's solutions to the above problems:

(i) To make sure that a connection comes to equilibrium, the sender uses the *slow-start* algorithm: Initially the window size is set to one; it is incremented by one, every time the sender gets an acknowledgment. This process continues until the window size has reached the maximum size agreed upon between the sender and the receiver at connection setup. In case of a timer interrupt in this phase of communication, the effective window size is dropped to one. For a window size of W, the slow start algorithm will normally take (in the absence of retransmissions) $log_2(W)$ steps to reach 'equilibrium'. Jain [Jai 86] had independently proposed a similar protocol called CUTE.

(ii) Once equilibrium is reached, the sender only transmits when it receives a previous ACK. The ACK of an old packet serves to *strobe a new packet* into the network. The conservation principle will now be violated only if the retransmit timer fails. In general, this timer is supposed to signal loss of a packet, but when the load becomes high, packets will be queued up at intermediate nodes, and this might cause the retransmit timer to post a premature interrupt, resulting in the sender retransmitting those same packets which are queued up in an already overloaded system. Jacobson's solution to this problem is an improved round-trip time estimator. Previous round-trip time estimators kept an estimate of the running mean of the round-trip time. Jacobson adds an estimator for the mean deviation of this time, and shows how his algorithm is able to better predict the round-trip time than the previous algorithm that was used in TCP (that one used a pre-determined constant unlike Jacobson's running estimate of the mean deviation). Now, with a good round-trip time estimator, a timer interrupt is most likely to imply a

packet loss.

(iii) Resource limits along the path: This is the most interesting (and complicated) problem that congestion control/avoidance seeks to alleviate. Similar solution strategies[3] have been attempted by Jacobson [Jac 88] and Jain, Chiu and Ramakrishnan [JCH 84, Jai 86, RaJa 88]. However, it is by no means solved in that nobody really knows how to adjust the window size.

Ramakrishnan and Jain [RaJa 88, 90] have implemented an *explicit feedback* mechanism from the congested node to the end-points when the congested node sees an average queue length of one. Their goal is to operate every node at the point where the global *power* (defined as throughput$^{\alpha}$/responsetime , [Klei 79]) is maximized. At that time, a bit is set in the outgoing packet so as to let the destination know about the congestion. The destination is responsible for quenching the source. This therefore, is a congestion-avoidance algorithm.

In an M/M/1 queue, power is maximized at a utilization, $\rho = 0.5$ (for $\alpha = 1$ in the power expression). For ρ equal to 0.5, we know that the expected queue length, $E[Q]$ is 1. The Ramakrishnan-Jain algorithm works as follows: When $E[Q] > 1$ is detected[4], all future packets in the current busy cycle are marked. The sources corresponding to these packets reduce their window size if at least 50% of their packets are marked.[5] To prevent wild oscillations

---

[3] The window size is decremented exponentially on congestion and incremented linearly otherwise. We shall discuss the details shortly.

[4] Obtained by averaging over the previous busy cycle and the current, incomplete one.

[5] The argument here is as follows. Suppose Q is the threshold queue length when the congestion indication bit is set. Let $p(n)$ be the probability of n packets at the node, including the one in service. Then the probability that the router sets a bit is $1 - (p(0) + p(1) + \cdots + p(Q-1))$. When $Q = 1$, this probability is equal to $1 - p(0) = \rho$ which is 1/2 for exponentially distributed service times. There are two approximations here, but both fortunately err on the conservative side: the relatively innocuous one is that the threshold at which a bit is set is really $E[Q] = 1$ over the last busy cycle and the current one and not $Q = 1$; the other is that when 50% of the bits are set, the variance in the estimate of congestion (or equivalently

in the window size, and to make sure that the feedback information is due to the value of the current window size, this change is performed at most once in two round-trip delays. Note however, that there may or may not be a correspondence between the sources whose packets are marked and those who are hogging the resource. Flow control using power as a metric is *not easily decentralizable* [Jaffe 90], but the statistical interleaving of packets may alleviate some of the unfairness.

Both [Jac 88] and [RaJa 88] suggest a 'multiplicative' decrease in window size on congestion detection, and then an 'additive' increase. That is, on congestion detection,

$$\text{window} \leftarrow \frac{\text{window}}{d}, \qquad d > 1 \tag{2.3.1}$$

The window size should grow back *slowly*. Both of them use

$$\text{window} \leftarrow \text{window} + a, a > 0 \tag{2.3.2}$$

Their choice of d and a is quite arbitrary. Jacobson chooses $d = 2$ and $a = 1$. The intuitive justification for $d = 2$ is the following: most of the time there is only one connection through a node. If a new connection also starts up, then the buffer should be equally divided. The justification of $a = 1$ is unfortunately not very convincing, even to the author of that paper. Ramakrishnan and Jain [RaJa 88] choose $d = 8/7$ and $a = 1$. They give reasons why a multiplicative decrease and an additive increase can achieve 'fairness' across all the connections running through that node. The values of d and a should determine the magnitude of oscillation of the window size and the time taken for the windows to converge to a fair value. The exact mathematical relationship has not been derived by them, however.

In our research, we have developed an approximate analytical model for this protocol. Our model is an extension of the Fokker-Plank Equation in

---

the error in that estimate) is also the highest. One is most certain of the condition of the queue when no bits are set or when all of them are. However, one is least certain of the congestion state when exactly half of them are set.

three dimensions: one is time, the other two are queue length and arrival rate. One fundamental difference in our model from the algorithms of Jacobson and Ramakrishnan-Jain is that we assume a rate based flow control instead of a window based scheme. Thus in our case, the control algorithm, $g(\lambda, q) = d\lambda(t,q)/dt$, is:

$$g(\lambda, q) = \frac{d\lambda(t,q)}{dt} = \begin{cases} C_0, & \text{if } q \leq \bar{q}, \\ -C_1\lambda, & \text{if } q > \bar{q}. \end{cases} \tag{2.3.3}$$

Here $\bar{q}$ is some arbitrary threshold value for the queue length. The analysis is discussed in detail in Chapter 6.

In the late 70's and early 80's, numerous other congestion control strategies were proposed. These protocols were based on selectively dropping packets based on hop count or input buffer limits, see for example [Irl 78, PeSch 75, SaSch 80, LaRe 79]. For a survey of these protocols, see the paper by Gerla and Kleinrock [GeKl 80]. These however, belong to a previous generation of protocols.

This completes our discussion of congestion control protocols.

Chapter 3

Evaluation of Error Recovery Protocols with Independent Packet Errors

3.1. Introduction

We start discussion by limiting the degrees of freedom to the case where (a) packet errors are *independent* and (b) the underlying network is a LAN (Local Area Network). These will be relaxed in the later chapters. As mentioned before, we are interested in quick response times for multi-packet message transfers. We shall evaluate the performance of the different retrasnsmission protocols over a local area network, characterized by low error rates, high bandwidth and low propagation delays.

Degradation of performance could result from a number of factors. It could be caused by flow control (for example, the outstanding window size could be very small), or by the host to network interface, or it could be caused by the choice of retransmission strategy in case of errors. Our focus here is on this last issue. The principle retransmission strategies that we consider are the *blast protocol with full retransmission on error (BFRE),* the *go-back-n* protocol, the *selective-repeat* protocol and the *optimal blast* protocol that we propose. Zwaenepoel [Zwa 85], presents an analysis of BFRE. He also presents *limited* simulations for the go-back-n and selective-repeat protocols, which suggest go-back-n as the strategy of choice for local area network environments. One of our contributions is the analytical evaluation of the go-back-n and selective-repeat retransmission strategies for a

multi-packet message. Our results corroborate those of Zwaenepoel: BFRE becomes unstable much faster with respect to message size than go-back-n or selective-repeat. However, BFRE has a very simple state machine and makes other design issues much simpler and efficient, see for example the network interface design of Kanakia and Cheriton [KaCh 88]. It also seems ideally suited for an environment where host processing time is a significant amount of the total time, precisely because the amount of "work" to be done by the host is reduced. This is the motivation for our *optimal blast protocol* which performs well for both large and small message sizes.

Previous analyses of go-back-n and selective-repeat assumed low nodal processing times, high error rates and high link delays [AnPr 86, BrMo 86, MQR 87]. The principal focus of those studies were on maximization of channel throughput, given assumptions of packet arrival rates and distributions. While that clearly was a viable goal for some environments, it is not the main focus for users interested in say, accessing files or making remote procedure calls over networks, where response times determine workstation performance. Towsley [Tow 79] had an interesting analysis of the go-back-n retransmission strategy, deriving formulas for *individual* packet delays under general assumptions of the distribution of packet arrivals at the sending site. This analysis would be more suitable for the nodes in store and forward networks.

Our study focuses on the statistics of the time to complete a multi-packet message transfer. We address both processing and transmission times. Most related work in this area, with the exception of [Zwa 85], ignore processing time as a negligible component of the delay. Measurements on local networks have shown that this delay is in fact significant.

The rest of this chapter is organized as follows. Section 3.2 presents the model and its assumptions and the protocol definitions. Sections 3.3 and 3.4 present the analyses of go-back-n and selective-repeat respectively. Numerical results comparing these protocols are presented in Section 3.5. We shall see that the performance of BFRE is very sensitive to message size. In Section 3.6, we propose and evaluate the Optimal Blast Protocol which

increases the range of operation of BFRE. Section 3.7 presents the analysis of go-back-n under the assumption that the transmission and processing times are generally distributed. Section 3.8 presents our conclusions and Appendix 3.A and 3.B fill in some of details omitted in Section 3.4.

3.2. Preliminaries

3.2.1. The Model

Figure 3.1 represents a typical network interface architecture. To transmit a packet, a station copies the data from host memory to interface memory and then transmits it onto the network.

When a packet arrives at a station, it is first put in interface memory from where it is copied to the host's memory. Messages are assumed to be comprised of fixed size data packets. The time to copy a data packet between host memory and interface memory is assumed to be a constant C. The time to transmit a data packet is assumed to be a constant T. The corresponding times for acknowledgment (ACK) packets are Ca and Ta respectively. Propagation delays are assumed to be negligible. C and Ca are limited by the DMA rate of the host bus. T and Ta are limited by the network's speed. In the analyses of Sections 3.3 and 3.4, we assume that there is just one send buffer. In case of multiple send buffers, the timing diagrams used in these analyses will change, but the method of analysis and the *relative* performance of the different protocols will not. In fact, we do generalize the analysis of go-back-n to handle arbitrary timing sequences. The focus here is on the relative performance of different retransmission schemes. We feel our analysis should be straightforward to extend to newer and faster interfaces.

Figure 3.2 shows the timing diagram of a simple sliding window protocol. We have assumed that the window size is large enough so that it does not close. The horizontal axis represents time. The upper, middle and lower

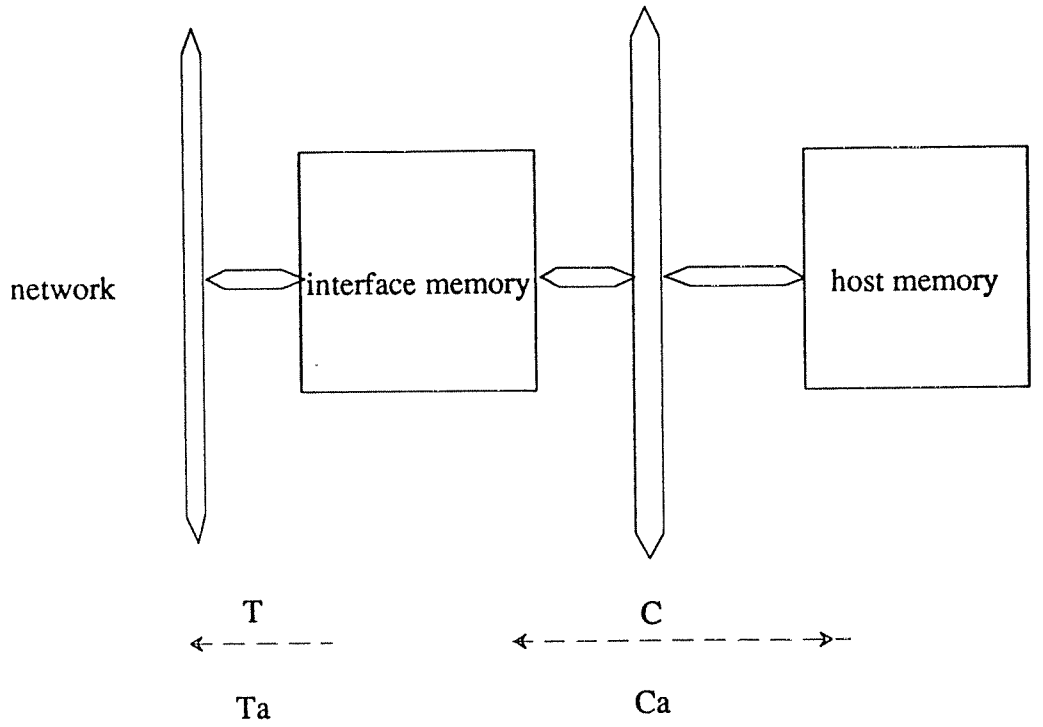

Figure 3.1: Network Interface Architecture

lines correspond to sending station, network and receiving station activity respectively. In this diagram, we show each packet being separately acknowledged. The sender first copies a packet from its memory to its interface. This takes C time units. The network transmission of this packet takes T time units. The data is then copied at the receiving end taking another C time units. Simultaneously, the sender transmits the next packet. Every packet is separately acknowledged. Copying of the ACK packet to the interface takes Ca time units and its network transmission takes Ta time units. Figure 3.3 shows the corresponding timing diagram of the Blast protocol. Here, the

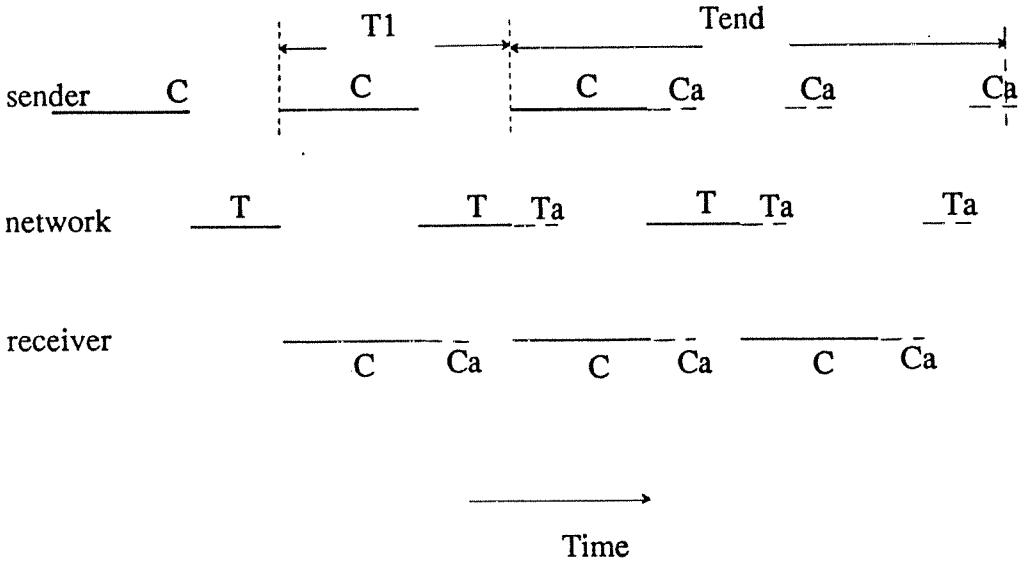

Figure 3.2: Timing Diagram of the Sliding Window Protocol (No Errors)

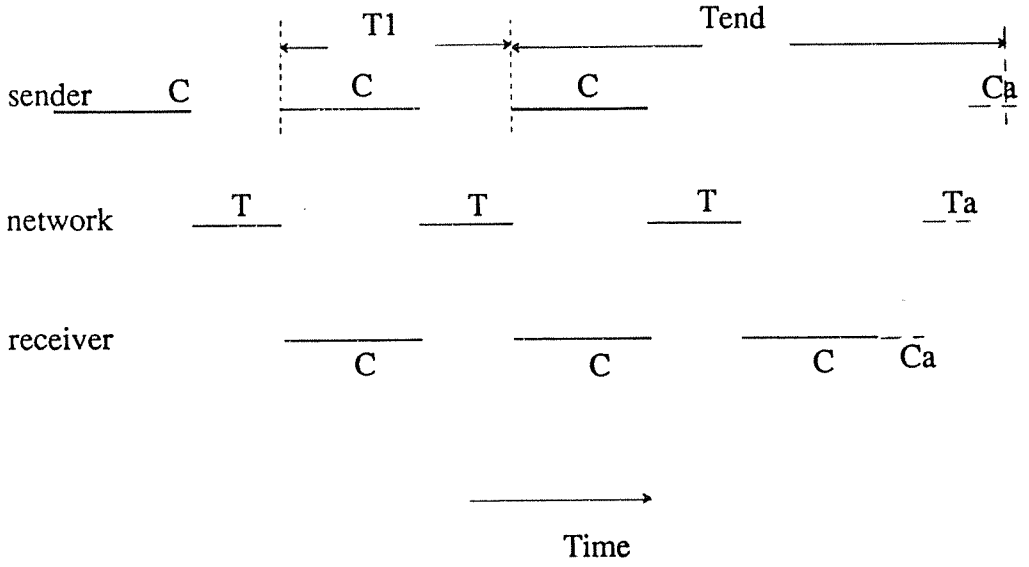

Figure 3.3: Timing Diagram of the Blast Protocol (No Errors)

receiver transmits an ACK only at the end of transmission of all packets.

In both these timing diagrams, it is assumed that there is one interface buffer for sending and one for receiving, and that the interface processes one packet at a time. This makes it possible, for example in Figure 3.2, for the the sender's data transmission to overlap with its processing of an

acknowledgment, i.e., data can be transmitted onto the network while an ACK packet is being copied into *host* memory. However, copying of data to the interface from the host cannot be overlapped with transmission of the data onto the network. The actual timing diagram will depend on the implementor's choice of signals and when they are masked off or turned on. It would also depend on the number of send buffers provided. However, the analysis we present in the next section would still remain valid if the time parameters chosen were suitably modified. In fact our analysis can be extended in a straightforward manner to the faster interfaces that are currently being designed [SoLa 88, KaCh 88].

The next important parameter of the model relates to packet error rates. Error rates in local networks are extremely low. If one out of every n bits are in error due to electrical noise, the probability of a packet of size b bits failing is $1 - (1 - 1/n)^b b/n + o(b/n)$. If data is transmitted as packets of 1K bytes each then the probability of a data packet failing is 8K/n. The corresponding packet failure rate for an ACK packet of say 64 bytes, is 512/n. For a bit error rate of one in 10^8 to one in 10^{10} or less, these values are extremely low. We are not aware of any authoritative report on the actual bit error rates on local networks. However, they seem to be sufficiently low, not to warrant any concern for performance degradation just by themselves (as we shall see in Section 3.5). The advent of optical fibers reduces errors to even lower rates. However, although collisions (in case of random access protocols) are rare, the increased use of remote file servers and other distributed applications are likely to increase their frequency. In addition, various studies [SoLa 88, Zwa 85] have reported significant error rates at network interfaces generally resulting from unavailability of buffers. Indeed Zwaenepoel suggests that packet error rates caused by interface errors are in fact somewhere in the range of one in 10^4 to one in 10^5 [Zwa 85]. Since this dominates network errors caused by random noise, we assume in our analysis that all packets have the same probability of failing, irrespective of packet size. This probability, which we denote by p_0, is an important parameter in our model. As we mentioned at the beginning of this chapter, we assume that these packet errors

are statistically independent, much as in [Zwa 85]. We shall see that this simplifying assumption actually helps shed some light on the performance of these protocols. However, this restriction will be removed in Chapter 5.

3.2.2. The Protocols

The protocols we are interested in are essentially retransmission strategies. We distinguish here between *transmission* and *retransmission* strategies. Briefly, the time when the receiver sends an ACK determines the transmission strategy (for example Blast and Sliding-Window are two different transmission strategies). A retransmission strategy, on the other hand, determines which packets are retransmitted in case of errors.

If the transmission strategy is sliding-window, the go-back-n and selective repeat retransmission strategies work as follows: when a packet successfully reaches the receiver, it is always ACKed if it is "in-sequence". In case of selective-repeat, the receiver buffers out of sequence data. In both cases an error is detected at the sender by either a timer interrupt or by a NACK from the receiver. At this point, if the sender backs up to the first packet in error and restarts the transmission, the strategy is referred to as go-back-n [Tan 81]. If, on the other hand, the sender retransmits only that packet which is in error, the strategy is called selective-repeat. In go-back-n, reassembling of the message at the receiver is much simpler than in selective-repeat, but at the potential cost of retransmission of many more packets.

The mechanisms for go-back-n and selective-repeat are similar if the transmission strategy is Blast. For a N-packet transfer, the first N-1 packets are transmitted unreliably (i.e., with no corresponding ACKs). The last packet is transmitted reliably, i.e., it is retransmitted periodically until an ACK is received. This ACK indicates the first packet in error in case of go-back-n, and all the packets in error in case of selective-repeat. The receiver also has a NACK capability to flag an error immediately when it is detected.

In BFRE, *all* the packets are retransmitted, irrespective of which packets

were in error. We have chosen to associate Blast as the transmission strategy with it. A sliding-window version with full retransmission seems to make less sense, because packets which have already been ACKed may then be (unnecessarily) retransmitted.

3.3. Go-Back-N Retransmission Strategy

In the go-back-n retransmission strategy, the sender retransmits all packets from the first packet in error. The receiver does not buffer out of sequence data. This simplifies the state machine, but at the potential cost of multiple retransmissions of successful packets. However, as we shall see, more sophisticated protocols cannot really improve on the performance of this protocol for realistic error rates.

3.3.1. Notation

We define the following symbols:

C : time to copy a data packet between host memory and interface memory

T : time to transmit a data packet onto the network

Ca : time to copy an acknowledgment (ACK) packet between host memory and interface memory

Ta : time to transmit an ACK packet onto the network

$T_1 : C + T$, time between the initiation of two successive data transmissions

$T_{end} : 2C + T + 2Ca + Ta$, time taken (as seen by the sender) to transmit the last packet and receive its acknowledgment.

τ : The time to detect an error at the sender *given that an error has occurred.* In Appendix 3.A, we have shown that for practical error rates, the variance of τ is very small in the presence of negative acknowledgments. We thus treat it as a constant here.

3.3.2. Analysis

This subsection presents the analysis of the expected time and the variance of the time to transmit N packets in the presence of errors. We assume deterministic processing times (C, Ca) and transmission times (T, Ta) and ignore queueing delays. We also assume that the sender can always send (i.e., if there is a window, it never closes), an assumption justified in light of our previous assumption of deterministic delays and no queueing.

Our analysis assumes a sliding window transmission scheme. A packet transmission fails when either the data packet or its corresponding acknowledgment is lost or is corrupted. Note that the failure of an acknowledgment does not necessarily mean a failed packet transmission, if for instance the acknowledgment for the next packet arrives before the sender times out. So this assumption overestimates the effect of an error and gives a lower bound on the performance of go-back-n. As stated in the previous section, we assume that packet failures are independent of their size and are also statistically independent. We denote the probability of packet failure by p_0. Given these assumptions, the probability that a packet transmission fails is:

$$p = 1 - (1 - p_0)^2$$

Now, suppose that the *first* failure occurs after r packets are successfully sent. The time to send the r packets and detect the error at the sender's site is :

$$T_f(r) = rT_1 + \tau, \qquad 0 \leq r \leq N - 1$$

where T_f indicates a failed transmission. For simplicity, we denote $q = 1 - p$. In go-back-n, the failure of a packet transmission marks a regeneration point of a stochastic process because all the packets starting from this point onwards have to be retransmitted. The probability of a regeneration occurring after r packets is $q^r p$.

The last packet sequence transmitted will have no errors. We denote the time for this transmission by $T_s(r)$, where r is the number of packets

transmitted in this last sequence.

$$T_s(r) = (r-1)T_1 + T_{end}, \qquad 1 \leq r \leq N$$

Its probability distribution is q^r.

Let the total time to successfully transmit N packets with the go-back-n strategy be T_N. If there are k regenerations (retransmission sequences), with r_i packets transmitted during the i^{th} retransmission, then the total time taken (denoted by $T(N|k)$) is :

$$T(N|k) = \sum_{i=1}^{k} T_f(r_i) + T_s(N - \sum_{i=1}^{k} r_i)$$

The above equation simplifies to

$$T(N|k) = (N-1)T_1 + T_{end} + k\tau$$

Let p_k be the probability that there are k regenerations given N packets. Since the last transmission always carries at least one packet successfully, the number of ways in which k regenerations can occur given N packets is $\binom{N+k-1}{k}$. To see this, note that this problem can be mapped to the problem of finding all possible integer solutions to the equation

$$X_1 + X_2 + \cdot + X_k + X_k + 1 = N$$

where $X_i \geq 0$ for $i = 1, 2, \cdots k$ and $X_{k+1} \geq 1$. Now, let $X'_{k+1} = X_{k+1} - 1$, so that $X'_{k+1} \geq 0$. Then the previous problem is analogous to finding all possible integer solutions to

$$X_1 + X_2 + \cdots + X_k + X'_{k+1} = N - 1$$

which is $\binom{N+k-1}{k}$. Then p_k is given by

$$p_k = \binom{N+k-1}{k} p^k q^N \tag{3.1}$$

The expected time to transmit N packets successfully is now easily obtained:

$$\begin{aligned} E[T_N] &= \sum_{k=0}^{\infty} T(N|k)p_k \\ &= [(N-1)T_1 + T_{end}] \sum_{k=0}^{\infty} \binom{N+k-1}{k} p^k q^N \\ &\quad + \tau \sum_{k=0}^{\infty} k \binom{N+k-1}{k} p^k q^N \end{aligned}$$

Now,

$$\sum_{k=0}^{\infty} \binom{N+k-1}{k} p^k q^N = (1-p)^{-N} q^N = 1$$

and

$$\begin{aligned} \sum_{k=0}^{\infty} k \binom{N+k-1}{k} p^k q^N &= q^N p \sum_{k=0}^{\infty} \frac{\partial}{\partial p} \binom{N+k-1}{k} p^k \\ &= q^N p \frac{\partial}{\partial p} \sum_{k=0}^{\infty} \binom{N+k-1}{k} p^k \\ &= q^N p \frac{\partial}{\partial p} (1-p)^{-N} \end{aligned}$$

and noting that $q = 1 - p$, this becomes $N\frac{p}{q}$ Thus $E[T_N]$ is given by

$$E[T_N] = [(N-1)T_1 + T_{end}] + \left[\tau N \frac{p}{q}\right] \tag{3.2}$$

Equation 3.2 has an obvious intuitive appeal. If $p = 0$, $E[T_N] = (N-1)T_1 + T_{end}$ is the time for an error free transmission (see Figure 2.2). For every failure, there is a cost of τ to detect the error. The average number of errors is the expectation of the distribution given by Equation 3.1 and is equal to $N\frac{p}{q}$.

We next compute the variance of the transmission time with the go-back-n strategy.

$$
\begin{aligned}
var(T_N) &= \sum_{k=0}^{\infty}(T(N|k) - E[T_N])^2 \binom{N+k-1}{k} p^k q^N \\
&= \tau^2 \sum_{k=0}^{\infty}\left(k - N\frac{p}{q}\right)^2 \binom{N+k-1}{k} p^k q^N \\
&= \tau^2 \left\{ \sum_{k=0}^{\infty} k^2 \binom{N+k-1}{k} p^k q^N - \left(N\frac{p}{q}\right)^2 \right\}
\end{aligned}
$$

Now, noting that $k^2 = k(k-1) + k$

$$
\begin{aligned}
\sum_{k=0}^{\infty} k^2 \binom{N+k-1}{k} p^k q^N = &\sum_{k=0}^{\infty} k(k-1) \binom{N+k-1}{k} p^k q^N \\
&+ \sum_{k=0}^{\infty} k \binom{N+k-1}{k} p^k q^N
\end{aligned}
$$

The first term on the right hand side can be derived in a manner similar to the derivation of Equation 3.2, except that we need to work with the second derivative now:

$$
\begin{aligned}
\sum_{k=0}^{\infty} k(k-1) \binom{N+k-1}{k} p^k q^N &= p^2 q^N \sum_{k=0}^{\infty} \frac{\partial}{\partial p} 2 \binom{N+k-1}{k} p^k \\
&= p^2 q^N \frac{\partial}{\partial p} 2 \left\{ \sum_{k=0}^{\infty} \binom{N+k-1}{k} p^k \right\} \\
&= N(N+1)\frac{p^2}{q^2}
\end{aligned}
$$

The second term is equal to Np/q, as derived before. These finally give

$$
var(T_N) = \tau^2 N \frac{p}{q^2} \tag{3.3}
$$

Equation 3.3 shows that the variance of the transmission time is proportional to the variance of the number of regenerations. The proportionality constant, τ^2, is small compared to the entire transmission time (see Appendix 3.A). Acknowledging every packet (or at least NACKing packets in error), reduces the time to detect an error. This is the only extra cost in go-back-n for each error.

A more complete analysis which accounts for variable transmission and processing times is given in Section 3.7. Assuming that T_1, T_{end} and τ are generally distributed i.i.d (independent and identically distributed) random variables with finite first and second moments, we find that the expected time and the variance of T_N are given by

$$E[T_N] = (N-1)E[T_1] + E[T_{end}] + E[\tau]N\frac{p}{q}$$

and

$$var(T_N) = (N-1)var(T_1) + var(T_{end}) + E[\tau]^2 N\frac{p}{q^2} + N\frac{p}{q}var(Td)$$

3.4. Selective Repeat

In this section we present the analysis of the selective repeat protocol. Several variations of this protocol have been proposed. Most assume that packet error rates are very high. Since this is not true in the LAN environment, we choose the following simple version. The sender transmits all N packets in the first *round*. The receiver sends an acknowledgment at the end of the round with a bit vector indicating the packets in error; these are retransmitted in the next round. This procedure continues until all packets have been successfully transmitted and received.

If there are k packets transmitted in a round, then the time taken is $kT_1 + T_{ohd}$, where $T_1 = C + T$ as before, and T_{ohd} is the overhead per round.

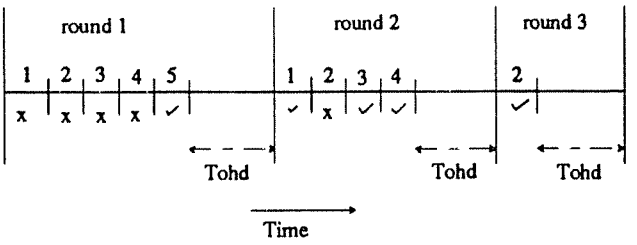

Figure 3.4: Selective Repeat, N = 5

We assume in the following analysis that the the sender always gets the ACK back after a time T_{ohd}. This assumption is strictly not necessary, but makes the results more intuitive and understandable. The analysis resulting from this simplification should favor selective repeat. Our main motivation in this section is to show that selective repeat cannot do very much better than go-back-n for practical error rates, so we choose to favor intuitive understanding over rigor.

To motivate the analysis, the reader is referred to Figure 3.4. We have broken the time line, as viewed by the sender, into *rounds*. In each round, all outstanding packets are transmitted. Correctly received packets are indicated by a tick while those requiring retransmission are indicated by a cross. The time line can be seen to consist of the sum of two random variables X and Y, where X is the sum of all the T_1's and Y the sum of all the T_{ohd}'s. The time to complete transmission of N packets is

$$T(N) = X + Y$$

and therefore,

$$E[T_N] = E[X] + E[Y]$$

and

$$var(T_N) = var(X) + var(Y) + 2cov(X, Y)$$

where $cov(X, Y)$ is the covariance of X and Y and is given by [Tri 82]

$$cov(X, Y) = E[XY] - E[X]E[Y]$$

The covariance term is not zero because the number of packet failures and the number of rounds are related (for example, the number of errors is at least equal to one less than the number of rounds).

3.4.1. Distribution of X

Each packet transmission takes a slot of duration T_1. Let us now consider a possible sequence of correct and erroneous transmissions which take $N+k$ slots (of size T_1 each), $k \geq 0$. Clearly, the $(N+k)^{th}$ slot is always a correct transmission. Hence, the total number of ways of distributing the k errors in $N+k-1$ slots is $\binom{N+k-1}{k}$. The probability of an error in a slot is $p = p_0$. Putting $q = 1-p$ as in section 3.3, we get

$$Pr[X = (N+k)T_1] = \binom{N+k-1}{k} p^k q^N$$

Therefore,

$$E[X] = T_1 \sum_{k=0}^{\infty} (N+k) \binom{N+k-1}{k} p^k q^N$$

which simplifies, much like Equation 3.2, to

$$E[X] = T_1 \frac{N}{q} \tag{3.4}$$

The variance of X is given by:

$$\begin{aligned} var(X) &= {T_1}^2 \sum_{k=0}^{\infty} \left(N + k - \frac{N}{q}\right)^2 \binom{N+k-1}{k} p^k q^N \\ &= {T_1}^2 \sum_{k=0}^{\infty} \left(k - \frac{Np}{q}\right)^2 \binom{N+k-1}{k} p^k q^N \end{aligned}$$

We know the result of this sum from the derivation of Equation 3.3:

$$var(X) = {T_1}^2 N \frac{p}{q^2} \tag{3.5}$$

3.4.2. Distribution of Y

For every round, we have a fixed overhead T_{ohd}. If there are R rounds then $Y = T_{ohd} * R$. Now, the distribution of R is given by

$$Pr[R \leq k] = \sum_{r_1=0}^{N} \binom{N}{r_1} p^{r_1} q^{N-r_1} \sum_{r_2=0}^{r_1} \binom{r_1}{r_2} p^{r_2} q^{r_1-r_2} \ldots \sum_{r_{k-1}=0}^{r_{k-2}} \binom{r_{k-1}}{r_{k-2}} p^{r_{k-1}} q^{r_{k-2}-r_{k-1}} q^{r_{k-1}}$$

which simplifies to

$$Pr[R \leq k] = (1 - p^k)^N \tag{3.6}$$

Viewed another way, since the total number of rounds is $\leq k$, each packet is transmitted successfully in at most k attempts. The probability of this event is $(1 - p^k)$. Since there are N packets, all of them encountering errors independently of each other, we get Equation 3.6. The expected cumulative overhead $E[Y]$ is now given by

$$\begin{aligned} E[Y] &= T_{ohd} E[R] \\ &= T_{ohd} \left[\sum_{k=0}^{\infty} Pr[R > k] \right] \\ &= T_{ohd} \left[\sum_{k=0}^{\infty} \left[1 - (1 - p^k)^N \right] \right] \end{aligned}$$

For $Np << 1$, this last expression can be approximated by

$$\begin{aligned} E[Y] &\approx T_{ohd} \left(1 + \sum_{k=1}^{\infty} Np^k \right) \\ &= T_{ohd} \left(1 + N\frac{p}{q} \right) \end{aligned} \tag{3.7}$$

The variance of Y is given by:

$$var(Y) = {T_{ohd}}^2 \sum_{k=1}^{\infty} \left(k - \left(1 + \frac{Np}{q} \right) \right)^2 * \left[\left(1 - (1 - p^k - 1)^N \right) - \left(1 - (1 - p^k)^N \right) \right]$$

Now using the formula for summation by parts, and assuming $Np << 1$, we can approximate this as follows:

$$\frac{var(Y)}{{T_{ohd}}^2} \approx \left(\frac{Np}{q} \right)^2 + \sum_{k=0}^{\infty} 1 \left[\left(k + 1 - \left(1 + \frac{Np}{q} \right) \right)^2 - \left(k - \left(1 + \frac{Np}{q} \right) \right)^2 \right] * \left(1 - (1 - p^k)^N \right)$$

$$\approx \left(\frac{Np}{q} \right)^2 + \sum_{k=1}^{\infty} \left[2k + 1 - 2 \left(1 + \frac{Np}{q} \right) \right] Np^k$$

and this finally yields

$$var(Y) \approx {T_{ohd}}^2 \frac{Np}{q} \left(1 - \frac{Np}{q} \right) \tag{3.8}$$

Equations 3.7 and 3.8 along with the covariance term from Appendix 3.B give the variance of the transmission time of selective repeat.

3.5. Numerical Results

This section compares the mean and variance of the transmission times of the go-back-n, selective-repeat and the BFRE protocols. The curves for BFRE are obtained from the analysis of [Zwa 85]. The results for go-back-n and selective-repeat are obtained from the derivations in Sections 3.3 and 3.4. We use the measured values of C, Ca, T and Ta reported in [Zwa 85] (Table 3.1). These values are getting progressively smaller with faster networks and interfaces, but we expect the relative times to be similar at least in the near future.

Parameter	Value
C	1.35 msec
Ca	0.17 msec
T	0.82 msec
Ta	0.05 msec

Table 3.1: Parameter Values

Figure 3.5 shows the expected time to transfer N packets for the different protocols, for $N = 64$ and $N = 512$. For $N = 64$, all three protocols have almost the same expected time for a packet error rate of 10^{-4} to 10^{-5} (the error range that we can expect in a local area network environment). As N increases, BFRE starts performing poorly. Go-back-n however fares almost as well as selective repeat even for $N = 512$.

An estimate of a parameter could be misleading without an estimate of its error. We therefore plot the standard deviation of the transmission times in Figure 3.6. The curves are for $N = 64$. The curve for BFRE assumes that the receiver has the NACK capability so that the sender can detect a failed transmission early. Go-back-n can be seen to have almost as low a standard deviation as selective-repeat for the error range of 10^{-4} to 10^{-5}. Selective-repeat does better for error rates of 10^{-2} and higher but that portion of the curve is not significant from a practical standpoint. The key point here is that go-back-n has a simpler state machine than selective-repeat and performs almost as well.

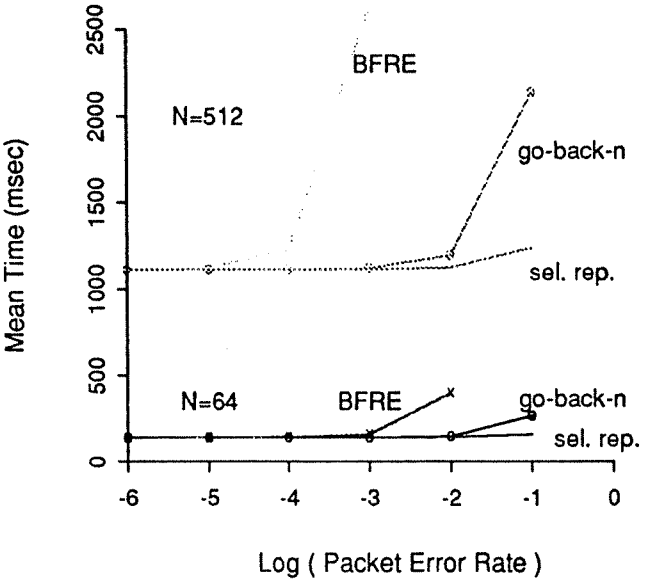

Figure 3.5: Mean time to transfer N packets versus packet error rate

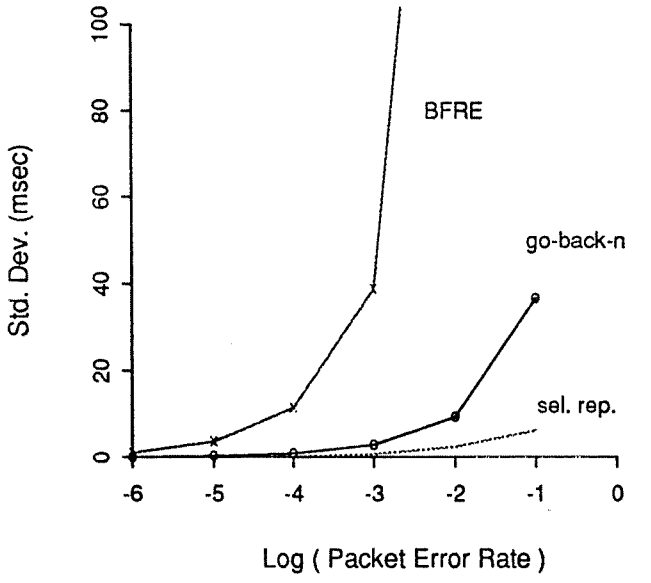

Figure 3.6: Standard deviation of the time to transfer N=64 packets versus packet error rate

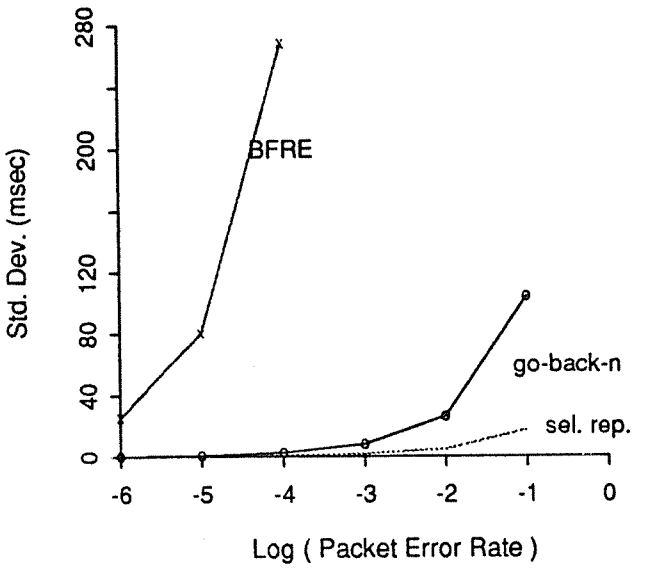

Figure 3.7: Standard deviation of the time to transfer N=512 packets versus packet error rate

In Figure 3.7, we have plotted the standard deviation curves for $N = 512$. This shows that even for large N, go-back-n is still a viable protocol. This figure clearly demonstrates that for large messages, the BFRE protocol, if adopted, should be decomposed into multiple BFRE's. We address this point in the next section in more detail. For large messages, we shall see that adding a checkpointing mechanism to BFRE at the right places is also a good alternative.

3.6. Optimal Blast Protocol

The Blast protocol with full retransmission on error (BFRE) is aesthetically simple and seems to take full advantage of the low error rates and high bandwidth of local area networks. However, its performance, especially the variance of the time to transfer large messages degrades considerably as message sizes increase. To avoid the performance penalties, without sacrificing much of the simplicity of the BFRE protocol, transmission of a large message can be decomposed into multiple BFRE's.

The number of packets in each BFRE could be *fixed apriori* or could be *variable*, with the latter enjoying the obvious advantages:

(i) Dynamic adjustability to changes in observed network error rates.

(ii) Tuning according to each individual sender's performance objectives.

The first point is obvious, especially if the error rates fluctuate with time (provided, of course, they can be estimated accurately). The second point emphasizes that the optimization criteria of different communicating pairs need not be the same. In the following discussion, we choose not to minimize the expected time to transmit a message because it is almost equal to the error free transmission time for practical error rates. Instead, we propose to constrain the standard deviation of the time to transmit the packets to some constant times the expected time to transmit the packets successfully. That is, the standard deviation, which we interpret as the error in the estimate of the mean, is constrained by the following equation:

$$\sigma(T_M) \leq rE[T_M], \qquad 0 < r < \infty \tag{3.9}$$

Typically, we would like r to have a very small value. Equation 3.9 says that we are less willing to accept large deviations for smaller messages than for larger messages. Also, we want the standard deviation to be smaller than some constant times the expected time to transmit the entire message. r serves as an upper bound on the coefficient of variation of T_M.

To achieve this desired standard deviation, for an M-packet-transfer, we propose to "checkpoint" the (blast) transmission by requiring a mandatory ACK from the receiver after every N packets, where N is the largest value such that Equation 3.9 is satisfied. This means that we have approximately M/N BFRE's in series, each of N packets. We call N the *optimal blast size.*

Let each BFRE be of size at most N packets. Let $n = M/N$. Then, ignoring the end effects of truncation and assuming that successive BFRE's are statistically independent, we have

$$var(T_M) = n\, var(T_N) \tag{3.10}$$

and

$$E[T_M] = nE[T_N] \tag{3.11}$$

The constraint in Equation 3.9 can then be rewritten as

$$n\,var(T_N) \leq r^2 E[T_M]^2 = r^2 n^2 E[T_N]^2 \tag{3.12}$$

Now, if the receiver NACKs on errors, [Zwa 85] shows that the variance of the time to transmit N packets using BFRE is

$$var(T_N) = t_0(N)^2 \frac{p(1+p)}{q^2} \tag{3.13}$$

where $t_0(N)$ represents the time to transmit N packets with no errors, p is the probability of a BFRE failing and $q = 1 - p$. The expected time to transmit the N packets is

$$E[T_N] = \frac{t_0(N)}{q} \tag{3.14}$$

From Equations 3.12, 3.13 and 3.14 we get

$$n \geq \frac{p(1+p)}{r^2}$$

and since $n \approx M/N$, we have

$$N \leq \frac{Mr^2}{p(1+p)} \tag{3.15}$$

The probability of a BFRE failing, p, is of course dependent upon N. It is the probability that at least one of the N packets that are transmitted fail, and is given by

$$p = 1 - (1 - p_0)^{N+1} \tag{3.16}$$

Given M, r and p_0, we can obtain N by solving Equations 3.15 and 3.16 iteratively to obtain the optimal blast size which satisfies Equation 3.9. Alternatively, when $Np << 1$, we have from Equation 3.16

$$p \approx (N+1)p_0$$

and therefore

$$N = \min\left\{\sqrt{\frac{M}{p_0}}, M\right\}$$

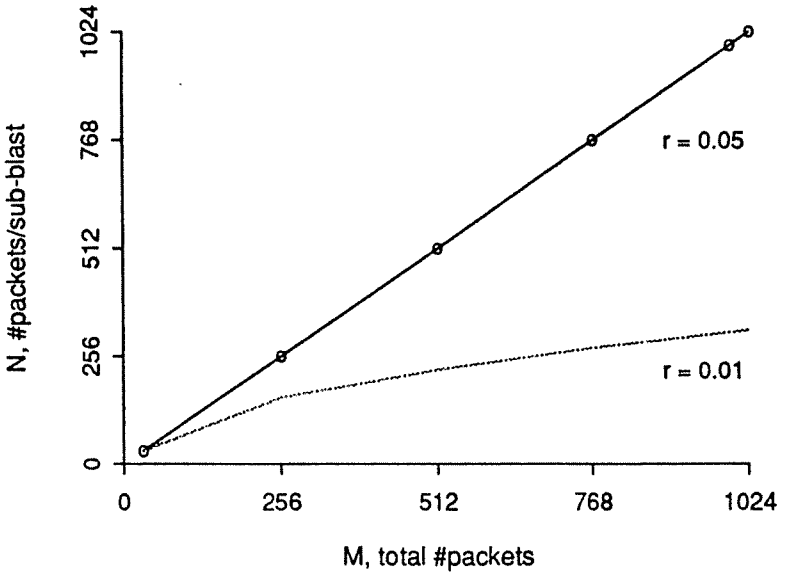

Figure 3.8: Optimal number of packets per sub-blast for $p_0 = 10^{-6}$.

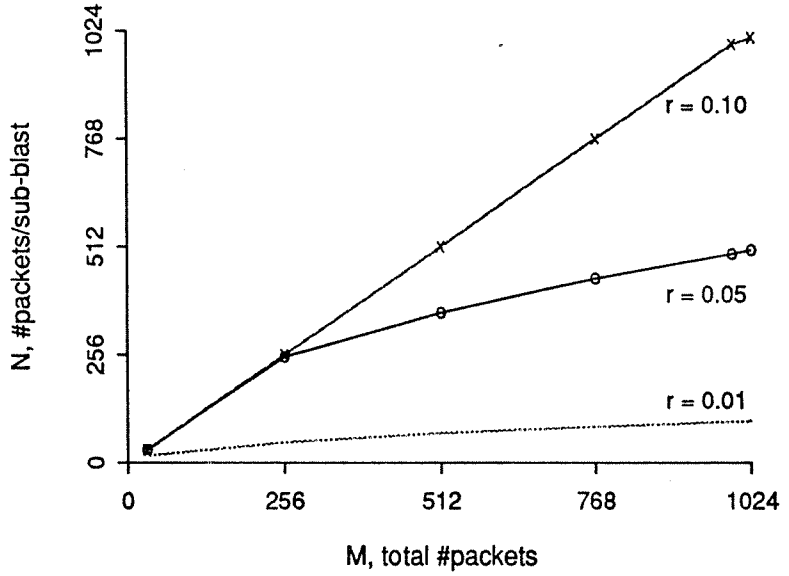

Figure 3.9: Optimal number of packets per sub-blast for $p_0 = 10^{-5}$.

In Figures 3.8 through 3.10, we show the optimal blast size for error rates between 10^{-6} and 10^{-4}, for different message sizes, M. Both the axes are in units of number of packets. It is interesting to see how the optimal blast size drops *rapidly* with increasing p and decreasing r. In Figures 3.11 and 3.12, we show a comparative performance of the optimal blast protocol

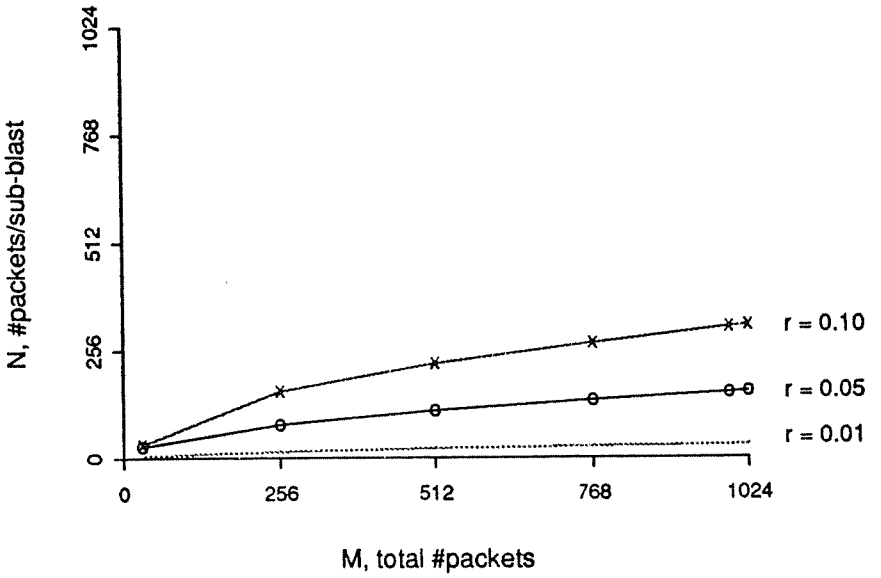

Figure 3.10: Optimal number of packets per sub-blast for $p_0 = 10^{-4}$.

and the normal BFRE protocol. The optimal blast protocol in these figures uses the optimal blast size for any particular M, r and p. In Figure 3.11, we have plotted the ratio of the expected times of the optimal blast protocol and BFRE. This value is close to unity. However, in Figure 3.12, we see the very sharp improvement in the standard deviation of the time, which essentially means that we have increased the confidence in the estimate of the mean almost for free. The reason is that the expected time is almost equal to the error free transmission time for practical error rates, but the standard deviation can still be large for large message sizes. We however see one problem with the optimal blast protocol: for small M, the ratio of the two expected times is greater than unity, especially as r gets smaller. This is because in our optimal blast, the sender waits for an ACK of the previous packet group before it starts transmitting the next packet group, causing the pipeline to empty out and fill up again for each sub-blast. The delay resulting from this dominates over the expected time of a simple BFRE for smaller message sizes because the probability of a retransmission is extremely low. Smaller values of r increase the number of sub-blasts (see Figure 3.10) exacerbating the problem. However, as M increases, one of the properties of

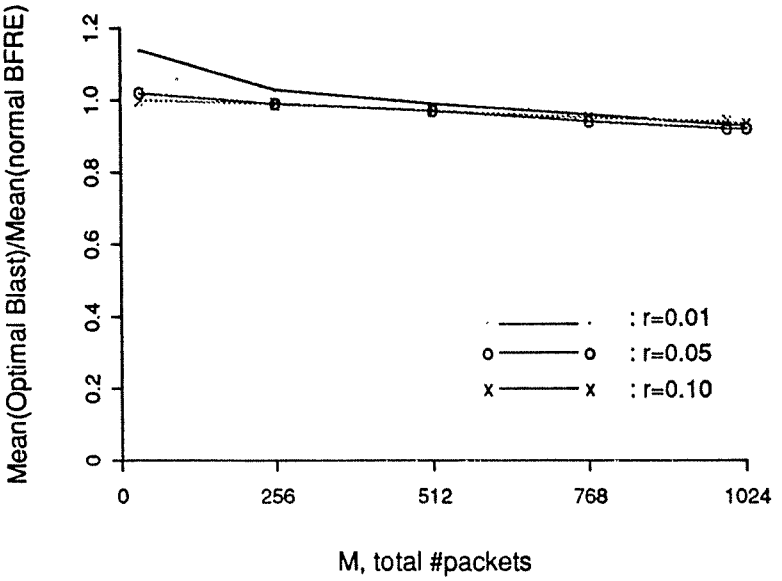

Figure 3.11: Ratio of expected time to transmit with optimal number of packets per sub-blast to ordinary BFRE. $p_0 = 10^{-4}$.

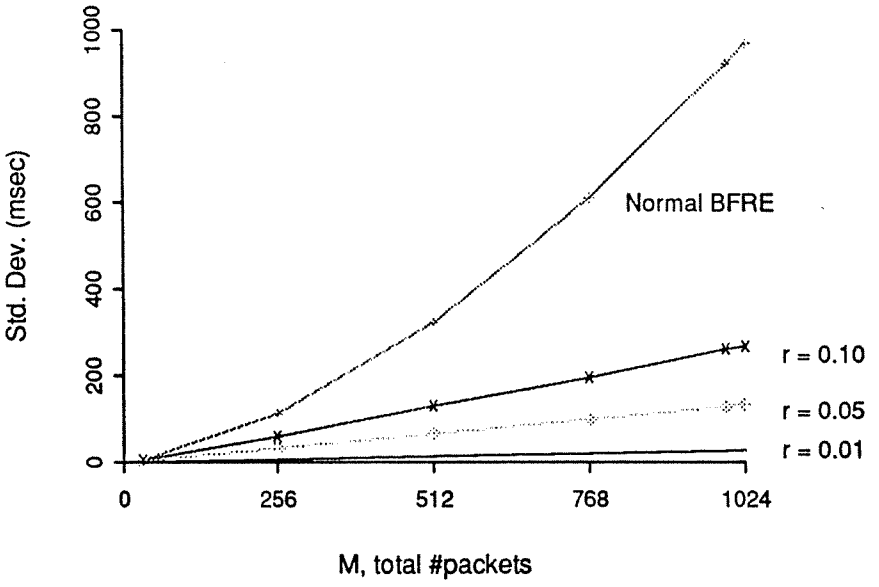

Figure 3.12: Std. deviation of transmission time with optimal number of packets per sub-blast. $p_0 = 10^{-4}$.

constraint 3.8 is that it increases the sub-blast size even though p_0 *and* r *are the same.* The pipeline does not empty out as often as before. In addition,

the probability of a retransmission increases for the simple BFRE. These factors pull the ratio of the expected times below unity as the total number of packets, M, increases. The standard deviation to the transmission time improves for all M, though it is more pronounced for large M.

To prevent the degradation in the expected transmission time for small M, we propose the following modification to the protocol:

(i) The sender determines the optimal blast size, N, for the given message.

(ii) It then transmits packets 1 through $N-1$ in the current BFRE without requesting an ACK from the receiver.

(iii) It transmits packet N with the REQUEST-FOR-ACK bit set.

(iv) Without waiting for the ACK, it continues with the next blast using steps (ii) and (iii).

(v) The receiver ACKs the packets which have their REQUEST-FOR-ACK bit set, provided it has received all the packets with sequence numbers greater than the previously ACKed packet and less than the current one. It can also NACK packets in error. Dropped packets however will have to be detected by the sender's timeout mechanism.

(vi) In case of an error (either a NACK or a timeout), the sender retransmits the whole "window" of outstanding BFRE's not yet ACKed. This leads to a go-back-n retransmission across sub-blasts, although each smaller sub-blast is still fully retransmitted!

We note that the sender does *not* have to negotiate the sub-blast size with the receiver in advance. In a window based flow control scheme, there has to be space for the packet when it arrives at the receiver, but flow control and error control are orthogonal functions here. One bit in the packet could serve as REQUEST-FOR-ACK/ NO-ACK, and could be set whenever the sender wants an ACK. Thus the size of a sub-blast could change with time even between the same communicating pairs. This could happen, for instance, if the sender's effective window size drops because it senses congestion. Jain and Jacobson [Jai 86, Jac 88] claim that packet errors are a good indicator of congestion, and their congestion control protocol shrinks the effective window size to deal with it . The window is slowly increased

after that. Their scheme fits in harmoniously with the sender's choosing the optimal blast size independent of the receiver. All that the sender has to do is to set the sub-blast size as min $\{N$, congestion-window, flow-window $\}$, where N is the optimal blast size from Equations 3.15 and 3.18.

3.7. Generalized Analysis of Go-back-n

We now generalize the go-back-n results by removing the deterministic time constraints under which the results were obtained in Section 3. We begin with some notation and definitions. Denote the time from the beginning of the transmission of packet i to the beginning of the transmission of packet $i+1$ by the random variable X_i, if the packet transmission was successful, i.e., both the data packet and its ACK were successful. This corresponds to T_1 in Section 3.3. The time corresponding to T_{end} is denoted as X_{end}. Thus X_{end} is a random variable denoting the time from the beginning of the transmission of the last packet until its ACK is received, given that the transmission is successful. Similarly let τ_i be the time to detect the ith error if one occurs. It is easy to see that the time to transmit N packets given that k regenerations have occurred is

$$T(N|k) = \sum_{i=1}^{N-1} X_i + X_{end} + \sum_{i=1}^{k} \tau_i, \qquad k = 0, 1, 2, \cdots$$

We assume that the X_i's are independent and identically distributed random variables with mean $E[X]$ and second moment $E[X^2]$. Also let their common Laplace transform be $X(s)$. Likewise we assume that $E[X_{end}]$, $E[{X_{end}}^2]$ and $X_{end}(s)$ are the mean, second moment and Laplace transform of X_{end}, and $E[\tau]$, $E[\tau^2]$ and $\tau(s)$ are the mean, second moment and Laplace transform of τ respectively (of course, we are assuming the τ_i to be i.i.d. random variables too). Then the Laplace transform of $T(N)$ which we denote by $T(s)$ is given by

$$T(s) = \sum_{k=0}^{\infty} \binom{N+k-1}{k} p^k q^N X(s)^{N-1} X_{end}(s) \tau^k(s)$$

$$= \frac{q^N X(s)^{N-1} X_{end}(s)}{[1 - p\tau(s)]^N} \tag{3.17}$$

Taking the natural logarithms of both sides of Equation 3.17, we get

$$ln(T(s)) = Nln(q) + (N-1)ln(X(s)) + ln(X_{end}(s)) - Nln(1 - p\tau(s)) \tag{3.18}$$

Notice that $E[X] = -\frac{d}{ds}X(s)\Big|_{s=1}$ and $E[X^2] = \frac{d^2}{ds^2}X(s)\Big|_{s=1}$, and similarly for the other random variables. Thus differentiating the left hand side of Equation 3.18, once and putting $s = 0$ gives $E[T_N]$ and differentiating it twice and evaluating it at $s = 0$ yields $var(T_N)$. The resultant equations are:

$$E[T_N] = (N-1)E[X] + E[X_{end}] + E[\tau]N\frac{p}{q} \tag{3.19}$$

and

$$var(T_N) = (N-1)var(X) + var(X_{end}) + E[\tau]^2 N\frac{p}{q^2} + N\frac{p}{q}var(\tau) \tag{3.20}$$

For the deterministic case in Section 3, $E[X] = T_1$, $E[X_{end}] = T_{end}$, $E[\tau] = \tau$ and $var(X) = var(X_{end}) = var(\tau) = 0$. As one would expect, the result is the same as given by Equations 3.2 and 3.3. Equations 3.19 and 3.20 are independent of the actual distribution of the $X_i's$ and $\tau_i's$, but depends only on their mean and variance. It is clear that the variance of the time to successfully transmit N packets will increase linearly with the variance of the protocol processing and transmission times and the time to detect errors. Also, Equations 3.19 and 3.20 are more general in the sense that they factor in various unaccounted for "random delays."

We do not have any real-life data on the variance of packet processing times and transmission times. In real implementations, there is likely to be a variation in packet processing times by the two stations. The variance of the transmission times could also be caused by network load, which, although usually low, can occasionally be quite high [Gus 87]. It is our surmise that packet processing and transmission times will be normally distributed about

their mean, but this needs empirical verification. Equation 3.20 is valid only if the random variables X_i, X_{end} and the τ_i's are independent of each other. It should apply to protocols implemented at the transport level or below, where correlations among consecutive packet transmission times are likely to be weak. The results of this section provide a means of isolating the communication of a pair of nodes from all other traffic. To some extent we have an expression for the mean and the variance of the delay for a bulk data transfer under a multiple-sender/multiple-receiver assumption. The results also apply to multiple hop transmissions, provided that windows never close at intermediate stations. The main problem that remains is to determine the mean and the variance of the $X_i's$ and X_{end}. The latter is likely to be more important as the number of hops increase and/or load from the other connections increase.

3.8. Summary and Conclusions

We presented analytical results for the expectation and the variance of transmission times for different retransmission strategies over local area networks. For small messages (i.e., small number of packets per message), BFRE, go-back-n and selective-repeat, all perform well. However, as the message size increases, BFRE shows larger mean and variance than go-back-n while the latter does almost as well as selective repeat. These conclusions are based on an estimate of the packet error rate between 10^{-4} and 10^{-5}. More reliable network interfaces will likely reduce error rates on local area networks. Under such conditions, BFRE will perform almost as well as the others, and given its simplicity, will be a more attractive protocol. For error rates which we observe today, go-back-n and the optimal blast protocol will be more viable alternatives since any protocol has to deal with a wide range of message sizes.

We also extended the analysis of go-back-n to handle the second order effects of variable processing and transmission times. We assumed a general

distribution of delays, instead of a deterministic one and showed how they affect the expected time and the variance of the transmission time of large messages. Possible applications of this model are datagram oriented transport protocols with associated protocol processing overhead, variable delays due to multiple connections, and variable transmission times due to network load. We found that for go-back-n the variance of a message transmission time increases linearly with the variance of individual packet transmissions in addition to that contributed by erroneous transmissions.

This study needs to be extended in many directions. We incorporate windows into our analysis in the next chapter. The effect of buffer non-availability at intermediate nodes (and the resultant correlated packet losses) is studied in Chapter 5. The effect of varying the transmission rates so as to reduce these packet losses is discussed in Chapter 6.

Chapter 4

Go-back-n with Windows

4.1. Introduction

In this chapter, we incorporate *windows* into the analysis of the *go-back-n* protocol. Previous studies have either been on flow control strategy or error control strategy *in isolation* [Mor 88, ToWo79]. The complexity of analyses has usually precluded a simultaneous study of both. Our main result in this chapter is that under certain circumstances, sliding window and go-back-n are *quasi-independent* in that they could be studied independently of each other and the results put back together in a straightforward manner. Thus, the window flow control protocol can be analyzed with models of varying complexity and then combined with the term representing the cost of errors.

This quasi-independence property is only an approximation, however. It is a good one for go-back-n but not for selective repeat. The rest of this chapter therefore, concentrates on go-back-n only.

4.2. Petri Net Models

Our goal is to show that sliding window flow control and go-back-n error control are quasi-independent. In the previous chapter, we had seen that this result was true when the window did *not* close, i.e.,

$$E[T_{N,gbn}] = E[T_{noErrors,N}] + \frac{Np}{q}\tau$$

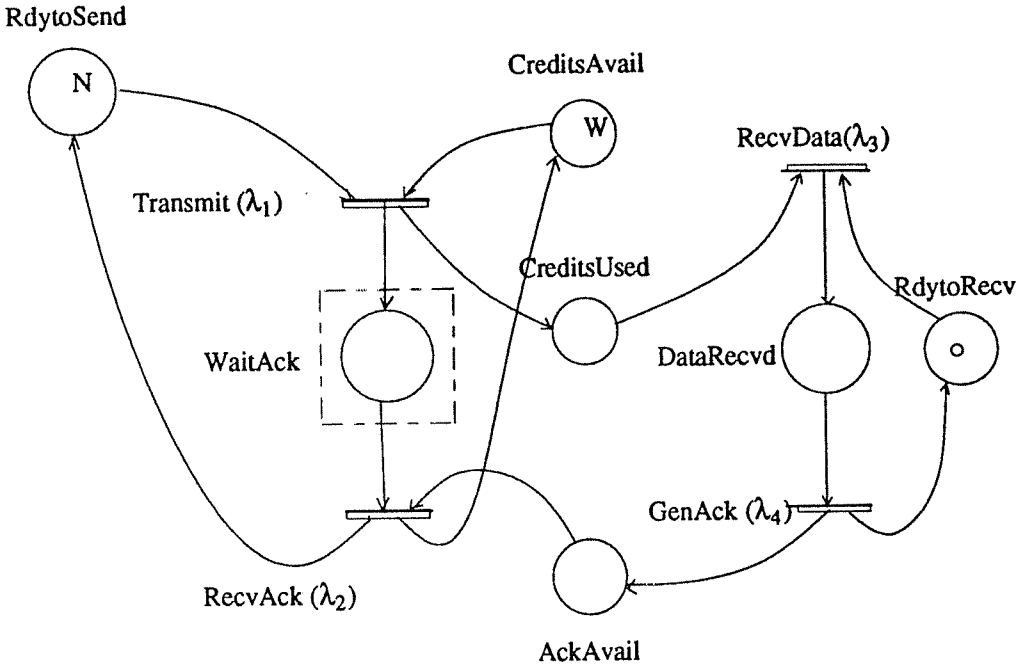

Figure 4.1: Simple sliding window flow control: Model I

where $E[T_{N,gbn}]$ is the expected time to transmit N packets in the presence of errors using go-back-n, $E[T_{noErrors,N}]$ is the time it would take to transmit N packets in an error free channel and $(Np/q)\tau$ is the extra cost due to errors, using go-back-n. We generalize this result here to the case when the window may close, *even with high probability.*

Figure 4.1 shows a Generalized Stochastic Petri Net (GSPN) model [MBC 84] of a simple sliding window flow control protocol ignoring all errors and retransmissions. If the place *RdytoSend* has a token, the sender can send a packet provided the place *CreditsAvail* has a token too. The mean time to send a packet is $1/\lambda_1$. When a packet is sent, one token from each of the above two places is removed; one is added to the place *WaitAck* where the sender waits for an acknowledgment. Another is added to the place *CreditsUsed* which is subsequently used by the receiver of the data. The transition *RecvData* can fire when the receiver has a token in *RdytoRecv* and a token is available in *CreditsUsed.* Upon receipt of the data, the receiver sends an acknowledgment packet which takes a mean time of $1/\lambda_4$. Notice

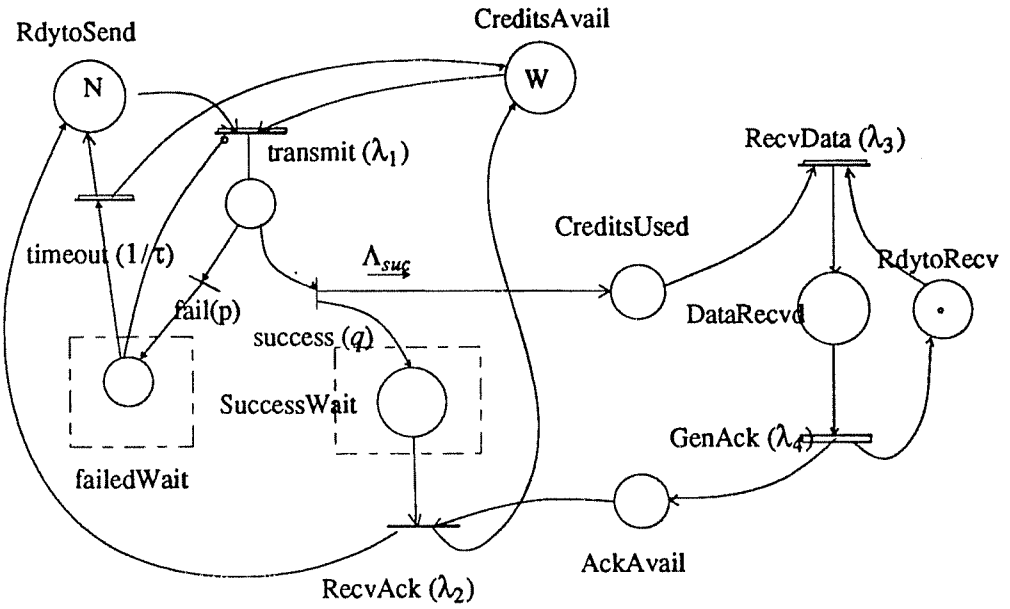

Figure 4.2: Sliding window flow control with go-back-n retransmission: Model II

that there are no errors in this model. For future reference, we shall call this Model I.

In Figure 4.2 we have the GSPN model of the same sliding window protocol but this time it includes the go-back-n retransmission strategy. In case of an error, all packets from the first packet in error are retransmitted. In the petri-net model, we suppress transmission of those packets which follow the erroneous one by using an inhibit arc from the *failedWait* place into the *transmit* transition. In a real implementation, these packets would actually have been transmitted (and then retransmitted). The inhibit arc, therfore, is an approximation because we are ignoring some of the additional loading effects at the receiver. The infrequency of these events should make this approximation reasonable.

A successful packet follows the same path as in Model I. In case of an error however, a token is deposited in the place *failedWait*. This inhibits further transmission at the sender. After a timeout interval of τ, the token in restored to the *RdytoSend* place and normal transmission begins.

A packet error could occur at different points in transit. Let the aggregate probability of error (of the packet or its acknowledgement) be p. In our

numerical examples later, we assume that both data and ACK packets have the same probability of failure, p_0, so that $p = 1 - (1 - p_0)^2$.

4.3. Analysis

4.3.1. Analysis of Model I

To study the effect of window size on throughput and round trip time of packets, we assume that the number of packets to be sent, N, is at least equal to the window size W, see Figure 4.1. This ensures that the sender always has a packet to send, and its transmission is delayed only if the window closes. Let $\overline{N}$ be the average number of tokens in the place *WaitAck*. Let ρ = Pr[CreditsAvail is not empty], which is the probability that the window is not closed. Then the throughput into the box marked with dashed lines is

$$\Lambda_1 = \lambda_1 \rho. \tag{4.1}$$

Let $\overline{R}_I$ be the average time spent by a token in the box. By Little's law, we have $\overline{R}_I = \overline{N}/\lambda_1\rho$, which implies that the expected number of packets initiated by the sender per round-trip time is $\overline{R}_I\lambda_1\rho$. The expected time to transmit N packets and receive the ACK for the last one, $E[T_{N,noErrors}]_I$, is given by

$$E[T_{N,noErrors}]_I = \overline{R}_I(\frac{N}{\overline{R}_I\lambda_1\rho}) + \overline{R}_I = \frac{N}{\lambda_1\rho} + \overline{R}_I \tag{4.2}$$

Since $\overline{N}$ and ρ can be computed using a Petri Net analyzer and $\overline{R}_I$ can be computed from $\overline{N}$, $E[T_{N,noErrors}]_I$ is easily obtained.

As before, let p be the aggregate probability of failure of a packet or its acknowledgment, and let $q = 1 - p$. It was shown in Chapter 3 that, if the windows *never* closed then

$$E[T_{N,gbn}] = E[T_{N,noErrors}]_I + N\frac{p}{q}\tau \tag{4.3}$$

where Np/q is the expected number of errors in go-back-n and τ is the expected cost per error. This result holds even for generally distributed processing and transmitted times. We shall show that Equation 4.3 holds approximately even when the window *may* close. We also present conditions under which this relation will be exact. Note that $E[T_{N,noErrors}]_I$ is computed from an *error free model.* The significance of this result is that we can actually analyze sliding window flow control and go-back-n error control as two simplified separate models and put the results back together in a simple way.

4.3.2. Analysis of Model II

In this sub-section, we present the analysis of $E[T_{N,gbn}]$ using the more detailed model in Figure 4.2. Let

Λ_{trans}= effective throughput through transition *transmit,*

Λ_{fail} = throughput through transition *failure,*

Λ_{suc} = throughput through transition *success,* and

r = $Pr[$ token in *failedWait* $]$.

Then, applying Little's Law to *failedWait,* we get $\Lambda_{fail} = r/\tau$, since τ is the expected time spent in the *failedWait* place. Now, noting that $\Lambda_{fail} = p\Lambda_{trans}$, we have

$$\Lambda_{suc} = q\Lambda_{trans} = (q/p)\Lambda_{fail},$$

which simplifies to

$$\Lambda_{suc} = \frac{r}{(\tau p/q)} \tag{4.4}$$

The average cycle time of a token in the successful path is obtained by applying Little's Law to the box around the place *SuccessWait :*

$$\overline{R}_{II} = \frac{\overline{N}(SuccessWait)}{\Lambda_{suc}}$$

The expected time to transmit N packets is then given by

$$E[T_{N,gbn}]_{II} = \frac{N}{\Lambda_{suc}} + \overline{R}_{II} \tag{4.5}$$

4.3.3. Comparison of the two methods

In Tables 4.1 through 4.6, we present the time to transmit 64 packets as calculated by the two models. We vary the parameters p, τ and W. We assigned measured values of $\lambda_1, \lambda_2, \lambda_3$ and λ_4 as reported in [Zwa 85]. Thus,
λ_1^{-1} = time to copy a data packet from the sending host's memory onto the wire = 2.17 msec
λ_2^{-1} = time to copy an acknowledgment packet from the wire into the sending host's memory = 0.17 msec
λ_3^{-1} = time to copy the data packet from the wire into the receiving host's memory = 1.35 msec and
λ_4^{-1} = time to copy an acknowledgment packet from the receiving host onto the wire = 0.22 msec

The time to complete an N-packet transmission is obtained by first solving the two GSPN models and then using their outputs as inputs to Equations 4.2, 4.3, 4.4 and 4.5. It can be readily seen that the time predicted by extrapolating Model I (in accordance with Equation 4.3) is remarkably close to that obtained by solving Model II (cf. columns 4 and 6 in Tables 4.1 - 4.6). This is in spite of the fact that the probability of the window closing or the probability of being in the *failedWait* state are not insignificant (see columns 2 and 5). We also vary p_0 from 10^{-2} to 10^{-5}, and τ from 10 to 1000 to show that this assumption is valid for a wide range of parameter values.

Let us now consider conditions under which the two models would be equal. Comparing Model I and Model II, we see from Equations 4.1, 4.2, 4.3 and 4.5, that the two models yield (asymptotically) identical results if

$$\frac{1}{\Lambda_1} + \frac{\tau p}{q} = \frac{1}{\Lambda_{suc}}$$

Model I				Model II	
W	$P[W=0]_I$	$E[T_N]_I$	$E[T_N]_I+\frac{Np}{q}\tau$	$P[failedWait=0]$	$E[T_N]_{II}$
1	0.4450	251.9	264.9	0.951	267.7
2	0.2337	183.5	196.5	0.933	198.6
4	0.0863	155.3	168.3	0.922	170.1
8	0.0172	145.9	158.9	0.917	160.9
16	0.0009	144.2	157.2	0.916	159.5

Table 4.1: N=64, $p_0=10^{-2}$, $\tau=10$

Model I				Model II	
W	$P[W=0]_I$	$E[T_N]_I$	$E[T_N]_I+\frac{Np}{q}\tau$	$P[failedWait=0]$	$E[T_N]_{II}$
1	0.4450	251.9	381.9	0.6607	384.7
2	0.2337	183.5	313.5	0.5851	315.5
4	0.0863	155.3	285.3	0.5420	287.0
8	0.0172	145.9	275.8	0.5249	277.8
16	0.0009	144.2	274.2	0.5217	276.4

Table 4.2: N=64, $p_0=10^{-2}$, $\tau=100$

Model I				Model II	
W	$P[W=0]_I$	$E[T_N]_I$	$E[T_N]_I+\frac{Np}{q}\tau$	$P[failedWait=0]$	$E[T_N]_{II}$
1	0.4450	251.9	1551.4	0.163	1554.2
2	0.2337	183.5	1483.0	0.123	1485.0
4	0.0863	155.3	1454.8	0.105	1456.5
8	0.0172	145.9	1445.3	0.099	1447.3
16	0.0009	144.2	1443.7	0.098	1445.9

Table 4.3: N=64, $p_0=10^{-2}$, $\tau=1000$

Model I				Model II	
W	$P[W=0]_I$	$E[T_N]_I$	$E[T_N]_I+\frac{Np}{q}\tau$	$P[failedWait=0]$	$E[T_N]_{II}$
1	0.4450	251.9	380.1	0.661	380.3
2	0.2337	183.5	311.7	0.585	311.8
4	0.0863	155.3	283.5	0.542	283.6
8	0.0172	145.9	274.1	0.524	274.2
16	0.0009	144.2	272.4	0.520	272.6

Table 4.4: N=64, $p_0=10^{-3}$, $\tau=1000$

Model I				Model II	
W	$P[W=0]_I$	$E[T_N]_I$	$E[T_N]_I+\frac{Np}{q}\tau$	$P[failedWait=0]$	$E[T_N]_{II}$
1	0.4450	251.9	264.7	0.951	264.7
2	0.2337	183.5	196.3	0.934	196.3
4	0.0863	155.3	168.1	0.922	168.1
8	0.0172	145.9	158.7	0.916	158.7
16	0.0009	144.2	157.0	0.915	157.0

Table 4.5: N=64, $p_0=10^{-4}$, $\tau=1000$

Model I				Model II	
W	$P[W=0]_I$	$E[T_N]_I$	$E[T_N]_I+\frac{Np}{q}\tau$	$P[failedWait=0]$	$E[T_N]_{II}$
1	0.4450	251.9	253.2	0.994	253.2
2	0.2337	183.5	184.8	0.992	184.8
4	0.0863	155.3	156.6	0.991	156.6
8	0.0172	145.9	147.1	0.991	147.1
16	0.0009	144.2	145.5	0.990	145.5

Table 4.6: N=64, $p_0=10^{-5}$, $\tau=1000$

For convenience, let us denote $h = \tau p/q$. Then for the previous condition to hold, we require that

$$\frac{1}{\lambda_1 \rho} + h = \frac{h}{r},$$

or

$$r = \frac{1}{1 + \frac{1}{\lambda_1 \rho h}} \tag{4.6}$$

Now, if the expected useful time spent per packet is t_{good} and the wasted time is t_{bad}, then from Model I and our quasi-independence hypothesis we have $t_{good} = \frac{1}{\lambda_1 \rho}$ and $t_{bad} = \tau p/q = h$. Equation 4.6 says that the expected times derived from the two models will be equivalent if

$$r = \frac{1}{1 + \frac{t_{good}}{t_{bad}}}$$

i.e.

$$Pr[failedWait = 1] = \frac{t_{bad}}{t_{good} + t_{bad}}$$

This would, by itself, make perfect sense *if* t_{good} *was somehow obtained from Model II.* We are, however, calculating $t_{good} = \frac{1}{\lambda_1 \rho}$ *from Model I* and r *from Model II.* The two models will be close if the probability that the window is open given that we are not in the midst of handling an error in the second model, is close to ρ, the probability that the window is open in the first model. The results from the petri-net analysis suggest that this is so.

4.4. Conclusions

In this chapter, we analyzed the go-back-n protocol in conjunction with sliding-window flow control. The analysis assumed that packet errors were independent of each other.

We discovered that go-back-n and sliding-window flow control are quasi-independent in that the total expected time to transmit an N-packet message

is approximately equal to the sum of the two separate results obtained by modeling each of them *independently of the other.*

A similar straightforward result does *not* exist for selective repeat, however. This is because the *cost per error* in selective repeat is *dependent* on the window size: if the window is always open for example, the cost of an error is just the time spent in transmitting the erroneous packet; if the window were to close on the other hand, the cost would depend on *which packet* in the current window failed and *what the value* of the window size was. The difficulty is not in being able to analyze this protocol with windows (we shall do it in Chapter 5), the problem is in obtaining a simple approximation similar to that of go-back-n (corresponding to Equation 4.3). That still is an open problem.

Chapter 5

Analysis of Error Control Protocols with Congestion-Dependent Errors

5.1. Introduction

In this chapter, we investigate the performance of go-back-n and selective-repeat error control protocols and their ability to recover from congestion loss. The performance measure of interest once again is the expected time and the standard deviation of the time to transmit a large message, consisting of N packets.

We first develop a framework to evaluate the two retransmission strategies in presence of windows when packet errors are congestion-dependent. As we noted in Chapter 1, earlier work on retransmission strategies [MLS 89, ToWo 79, Zwa 85], have assumed the independence of errors. If the cause of packet errors is random noise in the communications channel, then this is a reasonable assumption. However, in most networks, such random errors are extremely infrequent. A more common occurrence is packet losses at intermediate nodes due to lack of availability of buffers. When this happens, the premise of independent packet errors is no longer valid. In fact, it is more likely for a failure to occur when one has already occurred than when none has occurred. In our study, we assume first an abstract error model which can be used to represent any network error and then a more concrete one for a simplified system. We then compare the two retransmission strategies for different congestion models in the presence of window flow control.

We then are able to determine *when* an increase in window size can cause a sharp degradation in performance, and how the two retransmission strategies perform in such situations.

The rest of this chapter is organized as follows. Section 5.2 presents the two congestion models. In Section 5.3, we discuss the go-back-n protocol-model (with sliding window) and its analysis. Section 5.4 presents the same for selective repeat. Section 5.5 compares the two retransmission strategies with numerical examples and finally, we present our conclusions in Section 5.6.

5.2. The Model

We assume that the transmission time of a packet at the sender is exponentially distributed with mean $1/\lambda_1$. At the lower levels of protocol stack that we are interested in, the coefficient of variation of λ_1 is likely to be less than one. Our analysis is therefore only an approximation of real behavior. (However, since the mean of a sum of random variables is equal to the sum of their means, irrespective of their distribution, we expect our analysis to be rather accurate, at least with respect to the mean time to transmit the message).

The performance measures of interest are the statistics of the time to transmit a large multi-packet message consisting of N packets. The sender has a window of size w. This is the upper limit on the number of packets that it is allowed to transmit without waiting for an acknowledgment. The sliding window protocol, in conjunction with the go-back-n and selective-repeat retransmission strategies, works as follows. When a packet successfully reaches a receiver, it is always ACKed if it is 'in-sequence'. An error is detected at the sender by either a timer interrupt or by a NACK from the receiver. At this point, if the sender backs up to the first packet in error and restarts the transmission, the strategy is referred to as go-back-n [Tan 81]. If, on the other hand, the sender retransmits only that packet which is in error, the

strategy is called selective-repeat. The state machine of go-back-n is simpler than selective repeat. Also, the selective repeat protocol may require a large receive buffer to cache packets which are received correctly, but out of order. Go-back-n on the other hand, can operate with one receive buffer only. So, it is of interest to engineers and researchers to see if one can get away with this simple strategy.

In this chapter, we address congestion-dependent packet errors, i.e., the errors are caused by congestion in the communication channel. We develop two models for congestion-dependent errors, an *abstract* model and a *concrete* model. The abstract model represents any arbitrary network by a set of parameters. Careful choice of parameter values can yield useful insights on the *relative* performance of the two error control protocols. The concrete model is a first step towards a more detailed analysis of the innards of congestion. We have so far been successful in analyzing only a single node system. In the remainder of this section, we first discuss the abstract model and then the concrete model.

Abstract congestion model

We assume that if the current 'congestion state' of the system is α, then $p(\alpha)$ is the probability that a packet transmitted *now* will ultimately fail. We *explicitly* encode the information pertaining to the *current* transmission activity in α, much like in [BPU 88]. The details of the background network traffic and resource availability (or rather, un-availability) are however encoded by *implicit* parameters. Thus, for the selective repeat retransmission strategy, we assume $p(\alpha) = p(j,k)$, where j is the number of outstanding ACKs and k is the number of failures that have already taken place but not yet recovered from. For go-back-n on the other hand, all failures after the first one and before its detection are irrelevant. We therefore ignore the k-component and assume $p(\alpha) = p(j)$, where j is the number of packets with outstanding ACKs.

Let $w_{max} > w$, where w is any window size that we consider. Since $p(j,k)$ increases monotonically with both j and k, we may approximate it with an n-degree polynomial as follows: The j outstanding ACKs and k undetected failures could take away a maximum of $j+k$ buffers. In addition, k itself indicates the level of 'badness' of the congestion. Thus we may write

$$p(j,k) = p_0 + \sum_{i=1}^{n} a_i \left(\frac{j+k}{w_{max}}\right)^i + b_i \left(\frac{k}{w_{max}}\right)^i \tag{5.1}$$

where p_0 is the intrinsic failure rate of the network and the other terms are due to congestion. Note that the $a_i's$ and $b_i's$ are the implicit parameters representing resource un-availability due to congestion. If we increase the degree n in Equation 5.1, we can approximate any smooth curve more accurately. One possibility is to experimentally determine the curve for $p(j,k)$ by generating error statistics of a specific network. While that is a worthwhile study (and is work in progress), we can get important insights into the *relative* performance of go-back-n and selective-repeat by a careful exploration of the parameter space represented by the $a_i's$ and $b_i's$ in Equation 5.1.

The constants a_i and b_i are such that $0 \leq p(j,k) \leq 1$, i.e.

$$p_0 + \sum_{i=1}^{n} (a_i + b_i) \leq 1 \tag{5.2}$$

Concrete congestion model

Consider a single queue with a finite capacity, K. Let us assume that this system is fed by a 'background' stream of packets with exponentially distributed inter-arrival times and a 'foreground' traffic, which is our message of interest. Let us further assume that the service times are exponentially distributed. Then, with respect to the background traffic, this is an M/M/1/K queueing system. Let the background traffic intensity be ρ_b. Now consider

our designated message which arrives at this queue. Suppose that the packets of this message are spaced t_1 time units apart. A proper congestion control algorithm will attempt to make t_1 deterministic. (This reduces 'burstiness' and attempts to decrease buffer overruns). The question that interests us is: what are the *relative* probabilities of packet overflow for the sequence of packets of this message? (I.e., if there is a correlation, then what is it?).

The remainder of this section attempts to answer this question. First some definitions.

Let $p_k(m)$ = Pr $\{k$ customers in queue, including the one in service, when the mth packet of the message comes in $\}$.
Since our message arrives at a random point in time, the *first* packet sees the equilibrium probability distribution for buffer occupancy given a particular ρ_b. This distribution is given by [Kle 75]:

$$p_k(1) = \frac{1-\rho_b}{1-(\rho_b)^{K+1}}{\rho_b}^k \qquad 0 \le k \le K$$

Clearly, the probability of loss for the first packet due to the queue being full is $p_K(1)$. Now, if the retransmission strategy is *selective repeat,* we modify the probability distribution $\{p_k\}$ just *after* the packet arrives as follows:

$$p_k(\cdot) \leftarrow \begin{cases} p_{k-1}(\cdot), & \text{if } 0 < k < K; \\ 0, & \text{if } k = 0; \\ p_{K-1}(\cdot) + p_K(\cdot), & \text{if } k = K. \end{cases} \tag{5.3}$$

Essentially, this shifts the probability space one step to the right. We are then interested in the probability distribution when the next packet of this 'foreground' message arrives, t_1 time units later. This can be obtained by solving the Chapman-Kolmogorov equation for the Markov process (see [Kle 75]) : $d\mathbf{P}/dt = \mathbf{PQ}$, where $\mathbf{P}$ is the probability distribution matrix, $\{p_k\}$, and $\mathbf{Q}$ is the transition rate matrix of the queue and t is time. In our study, we used the *Uniformization* technique [Jens 53], to solve for $\mathbf{P}$ after a time t_1 given the current $\mathbf{P}$ as obtained by Equation 5.3. This method is summarized below:

Let q_{ij} denote the ith row and jth column entry of the matrix $\mathbf{Q}$. Let $q >= max_i|q_{ii}|$. Set $\mathbf{Q}^* = \mathbf{Q}/q + \mathbf{I}$. Then

$$\begin{aligned}\mathbf{P}(t) &= \sum_{i=0}^{\infty} \mathbf{P}(0)(\mathbf{Q}^*)^i e^{-qt}\frac{(qt)^i}{i!} \\ &= \sum_{i=0}^{\infty} \pi(i) e^{-qt}\frac{(qt)^i}{i!} \qquad (5.4)\end{aligned}$$

where

$$\pi(i) = \pi(i-1)\mathbf{Q}^*, \qquad \pi(0) = \mathbf{P}(0). \qquad (5.5)$$

We solve Equation 5.4 for $t = t_1$ with $\mathbf{P}(0)$ given by Equation 5.3. This gives the buffer occupancy probabilities just before the arrival of the next packet of our designated message, from where we get the buffer overflow probability, p_K. Repeated application of Equations 5.3 and 5.4 yield the probability of overflow of the subsequent packets. The solution process requires the computation of an infinite sum (see Equation 5.4). However, we found that we could easily truncate this sum because its tail goes to zero very quickly.

For *go-back-n* retransmission strategy, the algorithm given by Equation 5.3 is inappropriate. This is because a packet failure is relevant only if the previous packets have been successful. Thus, if packet m arrives at time t^-, then the distribution that is of interest at time t^+ is the one that will yeild the probability of loss for packet $m+1$. The probabilities computed at t^+ should be conditioned on the fact that packet m succeeded. The algorithm for modifying $p_k(\cdot)$ is therefore

$$p_k(\cdot) \leftarrow \begin{cases} \frac{p_{k-1}(\cdot)}{1-p_K(\cdot),} & \text{if } 1 \le k \le K\ ; \\ 0, & \text{if } k = 0; \end{cases} \qquad (5.6)$$

Repeated application of Equations 5.6 and 5.4 give the probability of overflow of consecutive packets when using go-back-n.

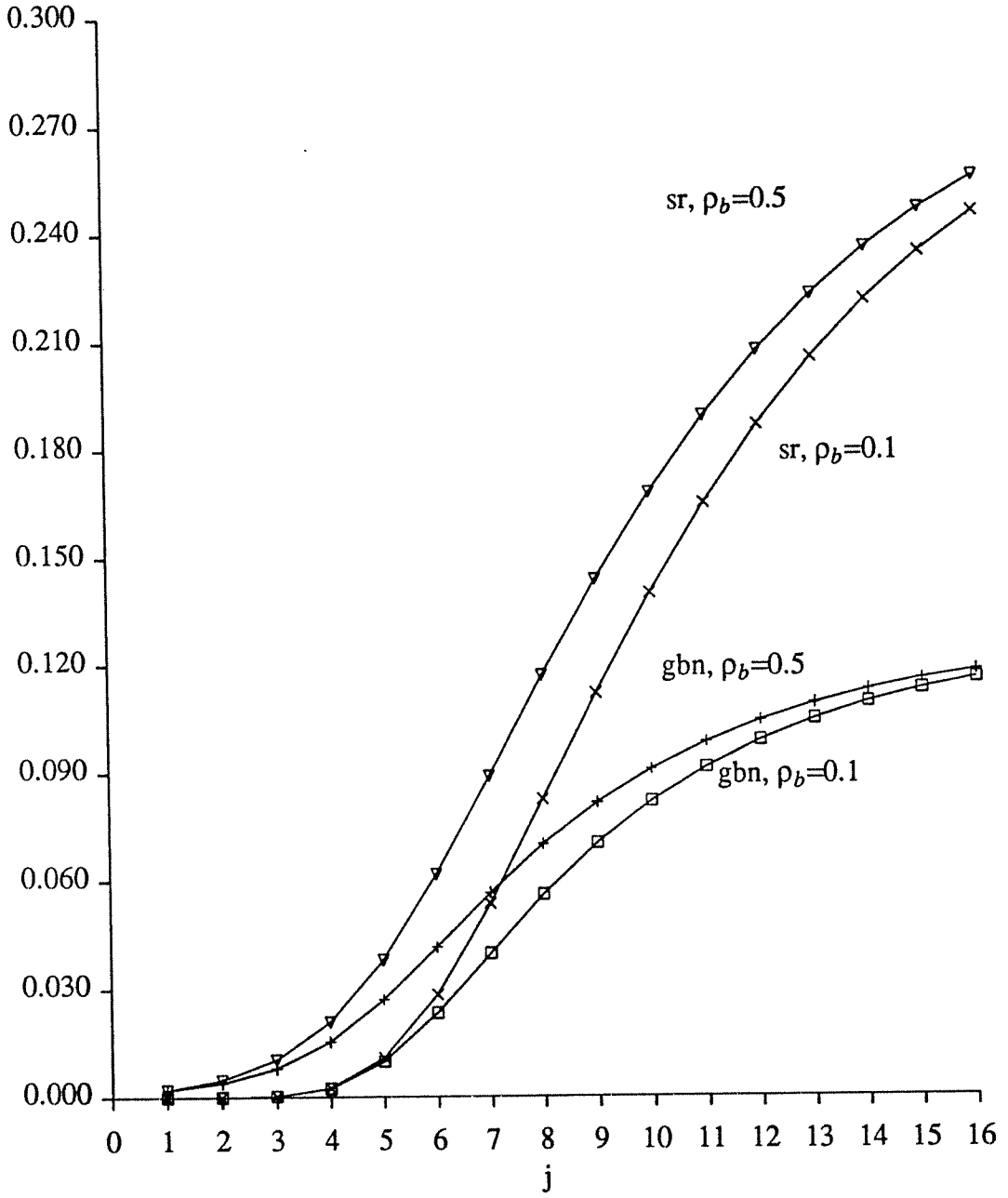

Figure 5.1: Probability of error in packet j for go-back-n and selective repeat $\rho_b = 0.1$ and 0.5, $t_1 = 1.0$.

In Figure 5.1, we plot some representative curves of correlated overflow probabilities for go-back-n and selective-repeat. Note that the selective-

repeat curves are above the go-back-n curves. This is however, *not* a disadvantage for selective repeat. The curve only says that the probability of a packet loss given that there were no previous losses is lower than that without any such conditioning. This is only to be expected because a packet that is transmitted after a previous one has been lost suffers a higher probability of loss. As long as that probability is less than one, however, the packet still has a chance to make it to the destination. In go-back-n, this limited chance is completely ignored. However, as we shall see later, it turns out that the probability of loss represented by the curves in Figure 5.1 (with or without the conditioning) are considerably high in that they degrade the performance substantially for *both* go-back-n and selective-repeat. The error curve for selective repeat, which represents a limited chance of getting through once congestion has set in, has therefore very little performance incentive as opposed to, say, a larger value of t_1 which pulls down the error curve (this is better *congestion* control). In this case however, go-back-n will perform almost as well as selective repeat, thus making it a viable protocol. It is then to be preferred over selective repeat if only on grounds of simplicity.

5.3. go-back-n protocol model

In this section, we present the protocol model for go-back-n. Simultaneously, we incorporate the congestion models that were presented in Section 5.2. Our goal is to derive the expected time to transmit an N-packet message using the go-back-n protocol with sliding window. To this end, a Continuous Time Markov Process is used to chart the progress of the message transmission. The state of the system consists of a pair of tuples (i, j) where i is the number of packets that will *not* require retransmission and j is the number of these i packets whose acknowledgments are still *outstanding*. Clearly $j \leq i$ and also $j \leq w$, if the window size is w. In addition, we also introduce states f_i corresponding to the states where an error has occurred after i packets have been successfully transmitted (see Figure 5.2).

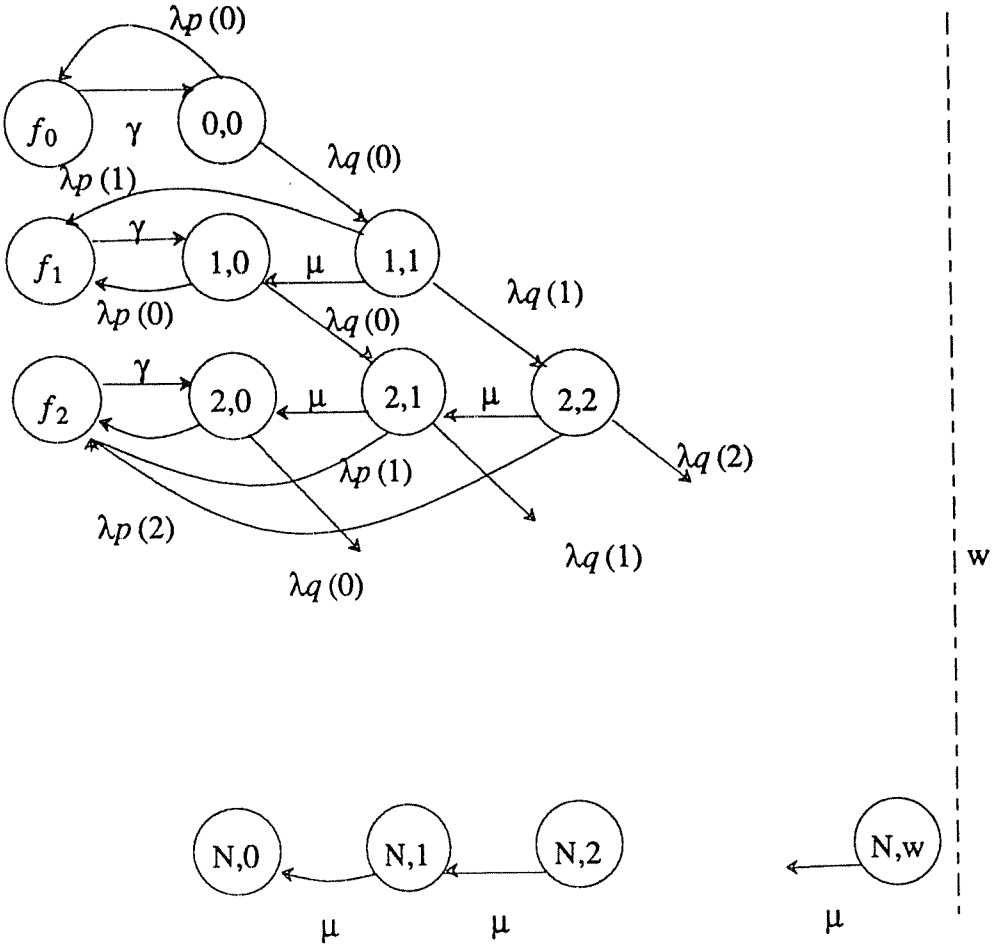

Figure 5.2: go-back-n with congestion-dependent errors and windows.

The sender transmits with a mean rate λ, and the acknowledgments return with a mean rate μ. Our hypothesis is that a packet fails with probability $p(j)$ in state (i,j), where j represents the level of congestion. The $p(j)$, $j \leq w$ are obtained from the congestion models of the previous section.

Figure 5.2 shows the state transition diagram of the ensuing Markov Process. The initial state is (0,0). When a packet is transmitted, there can be two possible next states. If the transmission is going to be successful (ultimately), we designate the next state as (1,1). Else, the packet transmission will fail and the next state is f_0. The rate into (1,1) is $\lambda q(0)$ and that into f_0 is $\lambda p(0)$. Once a packet fails, we assume that it is detected after a mean

time τ. Therefore, in Figure 5.2, we denote the rate from f_0 to (0,0) by τ^{-1} which we denote by γ. The rest of the arcs in the figure follow a similar argument. Note that for all j, a failure arc from (i,j) is into f_i and the recovery arc from f_i is only into $(i,0)$. This is a property of the go-back-n protocol: all the packets which are transmitted before a failure are represented by i. By the time the sender detects the failure of packet $i+1$ and acts upon it, the outstanding acknowledgments of all packets upto packet i must have returned to the sender for it to consider packet $i+1$ as the first failure and this then becomes the point of a new retransmission. Note that our model does not capture the congestion caused by those packets which were transmitted after a failed transmission but before its recovery. It is not difficult to add this information but we have not done so in this study for two reasons. First, if errors are caused by congestion, the timers should be relatively large so as to minimize the effects of congestion. This is in contrast to the independent-packet-error case where a timer tuned close to the roundtip delay is most desirable (see Equation 3.3, Chapter 3 for the performance implication of the timer). Second, the complexity of the solution process increases considerably. It may however, yield some insight into how long the timer value should be set so as to minimize the congestion effects of the previous packets. We shall explore this avenue in the near future.

Analysis

We next consider the transient analysis of the Markov Process of Figure 5.2. We set (0,0) as the initial state and (N,0) as the final state. We are interested in $E[T_N]$, the time to complete an N-packet transmission. However, $E[T_N]$ is just expected time to absorption into (N,0) for this Markov Process. To compute the expected time to absorption, we use the algorithm in [BRT 88]. Let η represent the vector of times spent in each of the states before absorption. Let $\mathbf{Q}$ be the transition rate matrix obtained from the original transition rate matrix by deleting the rows and columns involving

the absorbing states. Finally, let $\mathbf{P(0)}$ be the initial probability distribution of the non-absorbing states. Then the mean time spent in each state before absorption can be computed by solving for η [see BRT 88] in

$$\eta \mathbf{Q} = -\mathbf{P(0)} \tag{5.7}$$

The expected time to absorption is then given by

$$E[T_N] = \sum_{i,j} \eta_{i,j}$$

where $\eta_{i,j}$ are the individual components of η.

The solution of Equation 5.7 is especially simple for the Markov Process of Figure 5.2. For the states (0,0) and f_0, we have

$$\eta_{0,0} = \frac{1}{\lambda q(0)} \qquad \eta_{f_0} = \tau p(0)/q(0)$$

For other states (i,j), we get

$$\begin{aligned} \eta_{i,j}\left[\lambda 1_{\{j<w, i<N\}} + \mu 1_{\{j>0\}}\right] =& \lambda q(j-1)\eta_{i-1,j-1} 1_{\{i>0, j>0\}} \\ &+ \mu \eta_{i,j+1} 1_{\{j<i, j<w\}} \\ &+ \frac{1}{\tau}\eta_{f_i} 1_{\{j=0\}} \end{aligned} \tag{5.8}$$

$$\frac{1}{\tau}\eta_{f_i} = \lambda p(0)\eta_{i,0} 1_{\{i<N\}} \tag{5.9}$$

where

$$1_{\{C\}} = \begin{cases} 1, & \text{if C = true;} \\ 0, & \text{otherwise.} \end{cases}$$

Equations 5.8 and 5.9 are like 'flow equations', where we equate all the 'flows' into state (i,j) with all the 'flows' out of (i,j).

It turns out that for all states $(i,j), j > 0$ in level i, we have all the values needed to compute $\eta_{i,j}$, if we index through j from its highest possible value in state i downwards. Once these values are available, $\eta_{i,0}$ and η_{f_i} are given

in terms of each other and the other known values. This is a considerable simplification over using a general Gaussian elimination algorithm to solve Equation 5.7.

In Appendix 5.A, we present a method for determining the variance of the time to absorption. The expected time to absorption falls out of that analysis as a 'byproduct'. This helped us cross-check the numbers we obtained by solving Equation 5.7.

The solution to Equations 5.8 and 5.9 corroborates our previous results. For $p(j) = p \ \forall j$, we get $\eta_{f_i} = \tau p/q \ \forall i$. And if $w > i$, i.e. if the window does not close at the i^{th} level,

$$\sum_{j=0}^{i} \eta_{i,j} = \frac{1}{\lambda q}$$

So the expected time to transmit N packets is

$$\begin{aligned} E[T_N] &= N\left(\frac{1}{\lambda q} + \tau p/q\right) \\ &= N\left(\frac{1}{\lambda} + p/q\left(\frac{1}{\lambda} + \tau\right)\right) \end{aligned}$$

which is also a known result [MLS 89]. We can also use the Markov process to corroborate and somewhat strengthen our previous results for independent packet error for go-back-n with windows. In fact, it can be shown that

$$E[T_N, gbn] = E[T_N, noErrors] + O(p)$$

5.4. Selective repeat protocol analysis

In the Selective Repeat Protocol, the sender retransmits only those packets which are in error. We represent the state of a given transmission by the triplet (i, j, k) where i is the number of packets which have been successfully

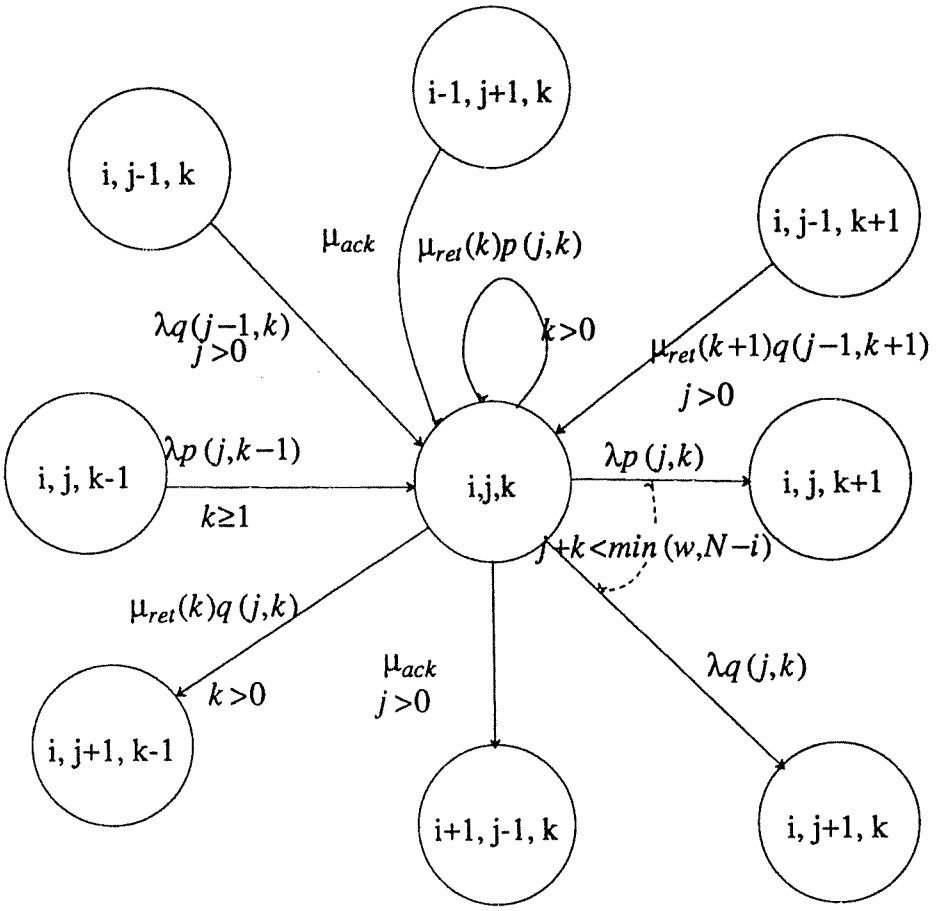

Figure 5.3: Selective Repeat state transition diagram.

ACKed, j is the number of (ultimately successful) packets whose acknowledgments are outstanding and k is the number of packets which have been transmitted but will fail and their failure is not yet detected by the sender. We assume that packet losses are more predominant than bit errors. Thus in state (i, j, k), we assume that the probability of a packet failing depends on j and k and we denote this probability by $p(j, k)$. Also, let $q(j, k) = 1 - p(j, k)$. We shall use the congestion models of Section 5.2 for $p(j, k)$. The 'abstract' congestion model poses no difficulty. For the 'concrete' model, we determine the probability of overflow assuming $j + k$ packets are outstanding.

An N-packet transmission starts off in state (0,0,0) and ends in state (N,0,0). Assuming the evolution of this process as a Continuous Time

Markov Process, we get the state transition diagram of Figure 5.3. To model a window of size w, we have the constraint $j + k \leq w$ for all states (i, j, k). If a new packet is transmitted from (i, j, k) (allowed only if $j + k < w$), the new state could be either $(i, j + 1, k)$ or $(i, j, k + 1)$ depending on whether or not this transmission will ultimately be successful. The corresponding rates are $\lambda q(j, k)$ and $\lambda p(j, k)$ respectively. If an acknowledgment comes back (with rate μ_{ack}) in state (i, j, k), the new state is $(i + 1, j - 1, k)$. If a failure is detected and the packet is successfully transmitted, the new state is $(i, j + 1, k - 1)$. We assume that the mean rate at which a packet error is detected in state (i, j, k) is given by $\mu_{ret}(k)$. This completes all the states to which a transition may occur from state (i, j, k). The states from which one may enter state (i, j, k) are shown in Figure 5.3 as a mirror image of the exit arcs. In the subsequent discussion, we drop the subscript ack from μ_{ack}.

One interesting property of the Markov process in Figure 5.3 is that no state may be visited more than once. To prove this formally, let us consider each of the possible exit states out of (i, j, k) separately. $(i + 1, j - 1, k)$ represents a state in which $i+1$ acknowledgments have already returned. We cannot ever get back from here to a state where there are only i successful acknowledgments. $(i, j + 1, k)$ and $(i, j, k + 1)$ represent a new transmission from state (i, j, k). A reduction from $j + 1$ to j in $(i, j + 1, k)$ will increase i. A reduction in $k + 1$ in $(i, j, k + 1)$ will increase j to $j + 1$ which will in turn increase i. Finally in case of a transition to $(i, j + 1, k - 1)$, a new failure will increase $k - 1$ to k giving $(i, j + 1, k)$, but then we have seen that $(i, j + 1, k)$ can never return to (i, j, k). This finally proves that state (i, j, k) can be visited at most once, i.e., the Markov process of Figure 5.3 is a directed graph with no cycles. This will help simplify the computation of the mean time to absorption, as we shall see shortly.

The rate of recovery from an error, $\mu_{ret}(k)$, satisfies the relation $\mu_{ret}(1) \leq \mu_{ret}(k) \leq k\mu_{ret}(1)$. The analogy here is to a 'First Come First Serve' scheduling of recoveries (the first inequality) and an 'Infinite Server' scheduling (the second inequality). To find the expected time to transmit N

packets, we solve for η in the equation [BRT 88]:

$$\eta \mathbf{Q} = -\mathbf{P(0)} \tag{5.10}$$

where η is the vector of expected times in each of the non-absorbing states, $\mathbf{Q}$ is the generator matrix obtained by deleting the absorbing states and $\mathbf{P(0)}$ is the initial probability distribution of the non-absorbing states. The expected time to absorption then is

$$E[T_N] = \sum_{i,j,k} \eta_{(i,j,k)}$$

Let us now proceed with the solution of Equation 5.10 for the Markov process of Figure 5.3. The equation for state (i,j,k) is given by:

$$\begin{aligned}
&\eta_{(i,j,k)}\left[\lambda 1_{\{j+k<w\}} + \mu 1_{\{j>0\}} + \mu_{ret}(k)q(j,k)1_{\{k>0\}}\right] \\
&= \eta_{(i,j,k-1)}\lambda p(j,k-1)1_{\{k\geq 1\}} \\
&\quad + \eta_{(i,j-1,k)}\lambda q(j-1,k)1_{\{j>0\}} \\
&\quad + \eta_{(i-1,j+1,k)}\mu 1_{\{i>0\}} \\
&\quad + \eta_{(i,j-1,k+1)}\mu_{ret}(k+1)q(j-1,k+1)1_{\{j>0\}} \qquad (5.11) \\
&\eta_{(0,0,0)} = 1/\lambda
\end{aligned}$$

where

$$1_{\{C\}} = \begin{cases} 1, & \text{if C = true;} \\ 0, & \text{otherwise.} \end{cases}$$

Equation 5.11 is like a 'flow equation', where we equate all the 'flows' into state (i,j,k) with all the 'flows' out of (i,j,k). Since each state is visited at most once, there are no cycles. Therefore if we begin with the 'root' of the directed graph and work outward, all the η's on the right hand side of Equation 5.11 will be available when required. The solution to Equation 5.10 can thus be obtained in a single pass.

In Appendix 5.A, we present a method for determining the variance of the time to absorption. The fact that the state transition diagram is a directed graph with no cycles helps reduce the complexity of that solution too, significantly.

5.5. Numerical Results

In this section, we compare the relative performance of the go-back-n and the selective repeat protocols when errors are dependent on congestion. The performance measure of interest is the expected time to transmit N packets. We also investigate the standard deviation of this measure to see how much confidence we can have on the expected value. If we set $p(\alpha)$ to be degree zero (cf. Section 5.2) we have $p(\alpha) = p_0$, which is independent of the congestion level and is hence the intrinsic packet error rate. In most networks, $p_0 \sim 10^{-5}$. In this case, we do not expect the relative performance of go-back-n and selective repeat to be very different, *cf.* Chapter 3.

To get the performance figures, we need the values of λ, μ, τ and $\mu_{ret}(k)$. We let the transmission rate, λ, be the same as that in Chapter 4, i.e., $\lambda = 1/(C+T)$. μ, in general, will depend on the window size w. We get its value from the Petri-net model of Chapter 4. This is only an approximation, because the μ obtained this way is a steady state value, whereas we are really interested in the the transient value of μ. However, we hope it would give a good indication of the *relative* performance of the two retransmission strategies, as the window size changes. Finally, we set $1/\tau = \mu_{ret}(1)$, and $\mu_{ret}(k) = k\mu_{ret}(1)$. This latter approximation may favor selective repeat somewhat. In our experiments, $1/\tau = \lambda/100$.

The interesting case with respect to errors is when they depend on the congestion level of the system. Therefore, we next consider $p(\alpha)$ to be of degree one, i.e., we let

$$p(j,k) = p_0 + a_1(j+k)/w_{max} + b_1 k/w_{max} \tag{5.12}$$

Here a_1 represents the effect of depletion of resources as the number of outstanding packets and their acknowledgments increase. A higher value of a_1 will correspond to a lower availability of buffers due to congestion. b_1, on the other hand, represents the decrease in service quality given that an error has occurred. Clearly, we expect b_1 to be much higher than a_1. This is because

W	$E[T_{N,gbn}]$	$E[T_{N,sr}]$ b_1=0.1	$E[T_{N,sr}]$ b_1=0.5
1	200.8	200.8	200.8
2	191.5	191.4	191.5
4	187.1	187.0	187.0
8	293.5	293.3	293.4
16	341.9	341.6	341.8

Table 5.1: Expected time to transmit N=64 packets. $a_1=10^{-4}$.

W	$E[T_{N,gbn}]$	$E[T_{N,sr}]$ b_1=0.1	$E[T_{N,sr}]$ b_1=0.5
1	200.8	200.8	200.8
2	222.4	215.3	223.0
4	293.2	264.1	295.2
8	591.7	471.4	544.3
16	722.7	608.6	716.2

Table 5.2: Expected time to transmit N=64 packets. $a_1=10^{-1}$.

W	$\sigma[T_{N,gbn}]$	$\sigma[T_{N,sr}]$ b_1=0.1	$\sigma[T_{N,sr}]$ b_1=0.5
1	20.5	20.5	20.6
2	20.0	19.5	20.3
4	21.1	20.3	21.6
8	38.4	37.4	38.5
16	45.5	44.1	45.4

Table 5.3: Standard deviation of the time to transmit N=64 packets. $a_1=10^{-4}$.

W	$\sigma[T_{N,gbn}]$	$\sigma[T_{N,sr}]$ b_1=0.1	$\sigma[T_{N,sr}]$ b_1=0.5
1	20.5	20.6	20.6
2	116.4	103.0	127.1
4	212.4	176.2	224.0
8	342.1	241.5	294.2
16	380.1	267.8	314.6

Table 5.4: Standard deviation of the time to transmit N=64 packets. $a_1=10^{-1}$.

once an error has occurred, we are more likely to be in an acute shortage of buffers, than otherwise.

Table 5.1 shows the expected time to transmit N=64 packets with go-back-n and selective repeat when $a_1 = 10^{-4}$ and b_1 takes values from 0 to 0.5. The effect of b_1 is seen to be negligible in this case, even for high values of b_1. This is because a_1 is so low that it is unlikely that the $j + k$ packets will have much effect on $p(j,k)$ when $k = 0$. Since $p(j,0)$ remains low (see Equation 5.12), the likelihood of hitting a state with $k > 0$ is very low, and so the effect of b_1 is negligible for this case.

Increasing a_1 does inflate the expected time, as we can see from Table 5.2, where we have put $a_1 = 10^{-1}$. The effect is more pronounced for larger window sizes as one would expect: the larger the window size, the larger the potential for congestion, and larger the potential for error. What is interesting, and not necessarily obvious, is the sharp degradation in performance as seen in Table 5.2. This is the network equivalent of thrashing. From Table 5.1, we note that the expected time decreases at first with respect to window size but then starts increasing again, implying that there is an optimum point for the window size. In Table 5.2, that optimum is for $w = 1$. Thus the optimum point of operating the window will change for different values of a_1. We are far from being the first to discover the potential for congestion as window size increases: Jacobson, Ramakrishnan and Jain, [Jac88, RJ88], have proposed dynamic window algorithms for the same purpose. Our contribution, however, is to quantify the effect of window size on the congestion level, and to corroborate the fact that larger windows do have a detrimental effect on performance when the network is congested (i.e., a_1 is high).

In Table 5.3 and 5.4, we tabulate the standard deviation of the time to transmit $N = 64$ packets for the same two values of a_1 as before. Notice that the standard deviation also gets worse with higher a_1, and this effect is again more pronounced for larger windows. A comparison of go-back-n and selective repeat shows that go-back-n performs roughly equal to selective repeat when $b_1 = 0.5$. One would normally expect selective repeat to perform better if b_1 is low, because that implies that an error does not significantly affect

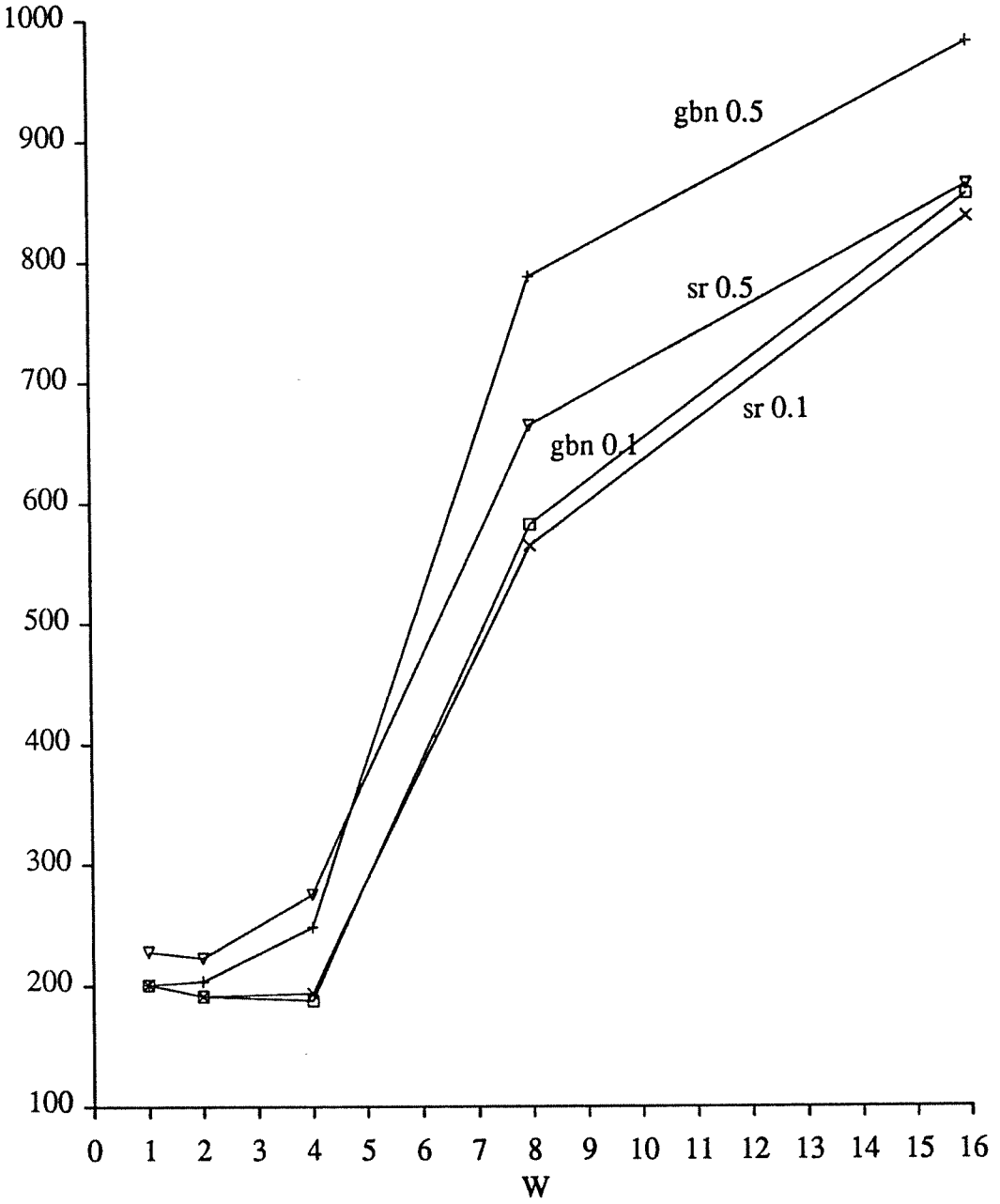

Figure 5.4: Expected time vs window size $N = 64$.
$\rho_b = 0.1$ and 0.5, $t_1 = 1.0$.

the 'state' of congestion. If, however, b_1 is high, transmitting more packets when an error has occurred can only worsen the congestion in the network.

Even selective repeat is seen to perform very poorly because associated with an error is the cost of detecting it. Retransmissions are therefore expensive.

Let us next consider the more concrete error model that we had discussed in Section 5.2. This was the single node model with finite buffers, carrying some 'background' traffic when a multi-packet message arrives. The packets in the message are assumed to arrive t_1 units of time apart. The parameters of this model are ρ_b, the background traffic intensity, t_1, the spacing between packets of the foreground message and K the number of buffers at this node. We let $K = 8$ in our experiments and vary ρ_b and t_1. Figure 5.4 shows the time to transmit $N = 64$ packets using go-back-n and selective repeat for $\rho_b = 0.1$ and 0.5 and $t_1 = 1.0$. Qualitatively, our conclusions are the same as before: when the system is thrashing (and losing packets), the performance is poor; while selective repeat may be slightly better, there is not much to write home about. On the other hand, when the system loses relatively few packets, go-back-n can match the performance of selective repeat.

5.6. Conclusions

In this chapter, we studied the go-back-n and the selective repeat protocols when packet errors were due to congestion loss. Specifically, we studied the effect of window size on performance.

We developed congestion models and protocol models to evaluate the two retransmission strategies. The performance measures that we considered were the expectation and the standard deviation of the time to completion of an N-packet transfer.

We considered two separate congestion models, an abstract one and a concrete one. In the abstract model, congestion dependent errors were a function of the current congestion level in the system. We denoted this by $p(\alpha)$ for a congestion state α. The choice of retransmission strategy may depend on this function. We tried some alternative functions for $p(\alpha)$ to determine how the two protocols compared. First we tried a function which

represented a low congestion level and then one which represented a relatively higher congestion level. We found that, irrespective of retransmission strategy, the expected time as well as the standard deviation of the time to transmit N packets increased sharply if the window size were large in the face of heavy congestion. This was the network equivalent of thrashing. We also saw the relative merits of the two retransmission strategies in these cases. If the congestion level was low, (cf. a_1 small in Section 5.5), the two retransmission strategies performed similarly. Under heavy congestion, it all depended on the value of the probability of back-to-back errors. Even if selective repeat was better, the difference was in the region where the performance was already *substantially* bad.

The concrete congestion model that we considered consisted of a finite queue with some background traffic level. We derived the correlated error patterns of a multi-packet message arrival when the packets of this message were separated by some predetermined deterministic interval. These probabilities were then used to drive the go-back-n and the selective repeat protocol models. The conclusions of this study was no different. The degradation due to large windows was much more pronounced, suggesting that flow control, and not retransmission strategy, is really the important issue under congestion.

Determining the congestion function $p(\alpha)$ is at the moment an open problem. It will depend on details of the system architecture like the number of buffers at each point in transit, the timing characteristics of incoming and outgoing links, the background traffic, etc. An experimental study using statistical techniques seems like a viable approach to determine the function $p(\alpha)$. We are pursuing this avenue.

The performance of congestion control strategies also needs to be investigated. In the next chapter, we present some of our results on the performance of dynamic congestion control strategies that are based on explicit or implicit feedback.

Chapter 6

Analysis of Dynamic Congestion Control Protocols

6.1. Introduction

In this chapter we investigate the performance of *congestion control protocols* that dynamically change input rates based on *feedback* information received from the network. This is motivated by recent proposals for adaptive congestion control algorithms [Jac 88, RaJa 88,90], where the sender's window size at the transport layer is adjusted based on perceived congestion level of a bottleneck node.

We develop, from first principles, a Fokker-Planck-like equation for the evolution of the joint probability density function of queue length and arrival rate at the bottleneck node. This approximates the *transient* behavior of a queue subjected to adaptive rate-control. We then seek answers to questions regarding *stability* (or oscillations) and *fairness* of a particular adaptive algorithm. We also investigate the effect of *delayed feedback* on performance.

We find that, in the absence of feedback delay, senders using the Jacobson-Ramakrishnan-Jain (or JRJ) Algorithm [Jac 88, RaJa 88,90] (or rather, an equivalent rate-based algorithm) *converge* to an equilibrium. Further, this algorithm is *fair* in that all sources sharing a resource get an equal share of the resource if they use the same parameters for adjusting their rates. The exact share of the resource that different sources get when they use different parameters is also determined.

A delay in the feedback information is shown to introduce cyclic behavior. If different sources get the feedback information after *different* amounts

of delay, then the algorithm may also be *unfair*, i.e., the sources may get unequal throughput. In a simulation study of the JRJ-protocol, Zhang observed oscillations in the queue length at intermediate nodes [Zha 89]. She also observed that connections with larger number of hops received a poorer share of an intermediate resource than those with a smaller number of hops. Jacobson also observed this independently in his measurements [Jac 88]. Our analysis not only concurs with these results, it also *explains* their *reasons*. The oscillations are due to delayed feedback; the unfairness is *partly* due to the larger (feedback) delay suffered by the longer connections as compared to the shorter ones.

The rest of this chapter is organized as follows. Section 6.2 presents the model. In Section 6.3, a Fokker-Planck approximation for the time dependent queue behavior is derived. Section 6.4 discusses the properties of the JRJ-algorithm when only one source is using the resource. Section 6.5 investigates the properties of the system with multiple sources. Section 6.6 re-investigates these properties in the presence of delayed feedback. Section 6.7 presents our conclusions.

6.2. Model

The model we have chosen is motivated by the Jacobson-Ramakrishnan-Jain Algorithm for window adjustment. In the JRJ algorithm, when congestion is detected (by implicit or explicit feedback), the window size is *decreased multiplicatively*. However, when there is no congestion, it is *increased linearly* — to probe for more bandwidth, i.e.,

$$w \leftarrow \begin{cases} w/d; & \text{if congested;} \\ w + a; & \text{if not congested.} \end{cases} \tag{6.1}$$

While this makes good intuitive sense, it is far from clear as to what *values* the parameters a and d should take. Further, it is not *provably* clear

if the algorithm is *fair* or *stable*[1] and if so, *under what circumstances.*

To understand the behavior of dynamic congestion control algorithms, we study a queueing system with a time varying input *rate.* The latter is adjusted periodically based on some feedback that the end-point receives about the state of the queue.

We are interested in the time evolution of the queue length density function. Let us assume that we are changing the arrival rate, $\lambda(t)$, based on the current queue length, $Q(t)$, at some bottleneck node. An example adaptive control algorithm could be

$$\frac{d\lambda}{dt} = \begin{cases} +C_0, & \text{if } Q(t) \leq \bar{q}, \\ -C_1\lambda, & \text{if } Q(t) > \bar{q} \end{cases} \tag{6.2}$$

where $\bar{q}$ is some target queue length. C_0 and C_1 are positive constants.

Equation 6.2 models a *linear increase* in λ for $Q(t) \leq \bar{q}$ and an *exponential decrease* in it for $Q(t) > \bar{q}$. It is therefore the rate-analogue of the dynamic window adjustment algorithm given by Equation 6.1. For purposes of generality however, we shall denote

$$\frac{d\lambda(\cdot)}{dt} = g(\cdot) \tag{6.3}$$

$g(\cdot)$ can be viewed as a generic rate-control algorithm.

To analyze the effect of Equation (6.2), Bolot and Shankar [BoSh 90] used two separate differential equations, one for the queue length, $Q(t)$, and another for the arrival rate $\lambda(t)$. They then coupled these two together by letting $\lambda(t)$ drive the differential equation for $Q(t)$ and vice-versa. This works fine when $Q(t)$ and $\lambda(t)$ are both *deterministic,* as is the case in their model. Suppose, however, that $Q(t)$ were a random variable and say, we were observing the process $\{(Q(t), \lambda(t))\}$ as time progressed. Given some initial values $(Q(0), \lambda(0))$, let the queue length at time t be $Q(t) = q$, for some q.

---

[1] An algorithm is *fair* if everybody gets a 'fair' share of the resource (*Fair* share and *equal* share are synonymous if all the demands are equal). Stability, on the other hand, implies that the algorithm converges to a particular value.

At this point, the value of $\lambda(t)$ is dependent on not just the current value of q, but also on the *sample path* of $Q(s), 0 \leq s \leq t$. Intermediate values of the queue length affects λ because of Equation 6.2 and since the sample path of Q is random, $\lambda(t)$ itself is a *random variable.* Coupling the two equations seems difficult now.

We therefore choose an alternate route. Let μ be the average service rate of the queue and let $\nu(t) = (\lambda(t) - \mu)$ be the instantaneous queue growth rate (with the convention that $\nu(t) = 0$ if $Q(t) = 0$ and $\lambda(t) < \mu$). We define $f(t, q, \nu)$ to be the joint probability density function of $(Q(t), \nu(t))$. Our goal is to understand the time dependent behavior of $f(\cdot)$ based on $g(\cdot)$ and the variabilities of $Q(t)$ and $\nu(t)$. We investigate this in the next section. The result is a Fokker-Planck like equation for $f(t, q, \nu)$.

6.3. Fokker-Planck approximation for queue with feedback control

Suppose that at time t, the queue length and queue growth rate are given by $Q(t) = \hat{q}$ and $\nu(t) = \hat{\nu}$. We want to express the density function $f(t + \tau, q, \nu)$ in terms of $f(t, \hat{q}, \hat{\nu})$. We assume that variability in ν is caused only by the random sample path of Q and there is no 'intrinsic' variability in ν. Then, given $Q(t + \tau) = q$, and some small τ,

$$\nu(t + \tau) = \hat{\nu} + g(\cdot)\tau. \tag{6.4}$$

Let $h(t + \tau, q, \nu | t, \hat{q}, \hat{\nu})$ be the conditional probability of the transition between $(\hat{q}, \hat{\nu})$ and (q, ν) in time time $(t, t + \tau)$. Then by the law of total probability,

$$f(t + \tau, q, \nu) = \int \int \frac{f(t, \hat{q}, \hat{\nu})}{1 + g_\nu \tau} h(t + \tau, q, \nu | t, \hat{q}, \hat{\nu}) \, d\hat{q} \, d\hat{\nu} \tag{6.5}$$

The integral over $\hat{\nu}$ in Equation 6.5 is a delta function which is zero for all values of $\hat{\nu}$ except that satisfying Equation 6.4. The factor $(1 + g_\nu \tau)$ preserves

the conservation of probability. We then have

$$f(t+\tau,q,\nu) = \int \frac{f(t,\hat{q},\hat{\nu})}{1+g_\nu \tau} h(t+\tau,q,\nu|t,\hat{q},\hat{\nu})\, d\hat{q} \tag{6.6}$$

with the understanding that $\hat{\nu}$ and ν are related by Equation 6.4.

Now, let us further assume that the central limit theorem (approximately) holds for the conditional density function $h(\cdot)$, i.e.,

$$h(t+\tau,q,\nu|t,\hat{q},\hat{\nu}) \approx \eta\left(\frac{q-\hat{q}-\hat{\nu}\tau}{\sigma/\sqrt{\tau}}\right) \tag{6.7}$$

where σ^2 is the variance of Q. Validity of this assumption is key to the Fokker-Planck approximation that follows.[2]

Combining Equations 6.6 and 6.7 gives

$$f(t+\tau,q,\nu) = \int \frac{f(t,\hat{q},\hat{\nu})}{1+g_\nu \tau} \eta\left(\frac{q-\hat{q}-\hat{\nu}\tau}{\sigma/\sqrt{\tau}}\right) d\hat{q} \tag{6.8}$$

To derive the differential equation of $f(\cdot)$ with respect to time, we subtract $f(t,q,\nu)$ from both sides, divide by τ and let $\tau \to 0$. We then get[3]

$$\begin{aligned} f_t &= \lim_{\tau\to 0} \frac{1}{\tau} \int \left\{ \frac{f(t,\hat{q},\hat{\nu})}{1+g_\nu\tau} - f(t,q,\nu) \right\} \eta(\cdot) d\hat{q} \\ &= \lim_{\tau\to 0} \frac{1}{\tau} \int \left\{ \frac{f(t,\hat{q},\hat{\nu}) - f(t,q,\nu)}{1-g_\nu\tau} \right\} \eta(\cdot) d\hat{q} - \int g_\nu(\cdot) f(t,q,\nu) \eta(\cdot) d\hat{q} + o(\tau) \\ &= \lim_{\tau\to 0} \frac{1}{\tau} \int \left\{ \frac{f(t,\hat{q},\hat{\nu}) - f(t,q,\nu)}{1-g_\nu\tau} \right\} \eta(\cdot) d\hat{q} - g_\nu(\cdot) f(t,q,\nu) + o(\tau) \end{aligned} \tag{6.9}$$

Let

$$I = \lim_{\tau\to 0} \frac{1}{\tau} \int \{f(t,\hat{q},\hat{\nu}) - f(t,q,\nu)\}\, \eta(\cdot) d\hat{q} \tag{6.10}$$

---

[2] higher order moments may be needed to express more burstiness in h.

[3] notation: $f_t = \partial f/\partial t$, $f_q = \partial f/\partial q$, $f_{qq} = \partial^2 f/\partial q^2$ etc.

Adding (and subtracting) $f(t,q,\hat{\nu})$ to (and from) the right hand side of this equation, we get

$$= \lim_{\tau \to 0} \frac{1}{\tau} \int \{f(t,\hat{q},\hat{\nu}) - f(t,q,\hat{\nu})\}\, \eta(\cdot) d\hat{q} + \frac{1}{\tau} \int \{f(t,q,\hat{\nu}) - f(t,q,\nu)\}\, \eta(\cdot) d\hat{q} \tag{6.11}$$

The first integral in Equation 6.11 is the Fokker-Planck equation [New 68, New 71, Kle 76]:

$$-\hat{\nu} f_q(t,q,\hat{\nu}) + \frac{1}{2}\sigma^2 f_{qq}(t,q,\hat{\nu})$$

As $\tau \to 0, \hat{\nu} \to \nu$, (see Equation 6.4), so this becomes

$$-\nu f_q(t,q,\nu) + \frac{1}{2}\sigma^2 f_{qq}(t,q,\nu) \tag{6.12}$$

The second integral is equal to

$$\begin{aligned} &\lim_{\tau \to 0} \frac{1}{\tau} \{f(t,q,\nu - g(\cdot)\tau) - f(t,q,\nu)\} \\ &\qquad = \lim_{\tau \to 0} \frac{1}{\tau} \{[f(t,q,\nu) - g(\cdot)\tau f_\nu(t,q,\nu) - f(t,q,\nu)] + o(\tau)\} \\ &\qquad = -g(\cdot) f_\nu(t,q,\nu) \end{aligned} \tag{6.13}$$

Combining Equations 6.9, 6.10, 6.11, 6.12 and 6.13, and noting that $g_\nu f + g f_\nu = (gf)_\nu$, we have

$$f_t + \nu f_q + (gf)_\nu = \frac{1}{2}\sigma^2 f_{qq} \tag{6.14}$$

Equation 6.14 describes the basic equation of motion for the density function $f(\cdot)$.

6.4. Properties of Algorithm 6.2

We now investigate the properties of Algorithm 6.2 in conjunction with Equation 6.14. For the purposes of an intuitive discussion, we suppress the σ^2 term in Equation 6.14 and study a reduced system. We therefore have a hyperpolic partial differential equation whose properties can be explored by studying its *characteristics*. Consider the $q - \nu$ diagram of Figure 6.1. The x-axis represents the queue length, Q, and the y-axis represents the instantaneous queue growth rate, ν. Two lines corresponding to $Q = \bar{q}$ and $\nu = 0$, shown by dotted lines, divide the $q - \nu$ plane into four quadrants. The behavior of Equation 6.14 is best described by considering each quadrant separately.

First consider Quadrant I in Figure 6.1. This corresponds to $\nu > 0$ (i.e., $\lambda > \mu$) and $Q < \bar{q}$. Since $\lambda > \mu$, the *instantaneous* queue length at any point in this quadrant is increasing. The instantaneous ν is also increasing because $d\lambda/dt = C_0 > 0$. The *resultant* direction of instantaneous motion (i.e., the characteristic) is increasing in both Q and ν as shown in the figure. Notice that Equation 6.14 confirms this intuition: the coefficient of f_q which represents the *Q-drift* is ν and this is positive in Quadrant I; the coefficient of f_ν which represents the *ν-drift* is $g(\cdot) = +C_0$ which is positive as well. The *characteristic* is the resultant of these two drifts.

Next, consider Quadrant II. Here $Q > \bar{q}$ and $\nu > 0$ (i.e., $\lambda > \mu$). From Equation 6.14, the Q-drift is again positive since $\nu > 0$. However, the ν-drift is now negative because $d\lambda/dt$ is $-C_1\lambda$ for $Q > \bar{q}$. The characteristic, which is the resultant of these two drifts, is increasing in Q but decreasing in ν as shown in Figure 6.1.

We can similarly check that in Quadrant III, both the Q-drift and the ν-drift are negative while in Quadrant IV, the Q-drift is negative but the ν-drift is positive. The directions of individual drifts and the characteristics are shown in the figure.

Now, suppose we were to trace the path of a 'particle' that obeys both Equation 6.14 and Equation 6.2. This path will follow the characteristic.

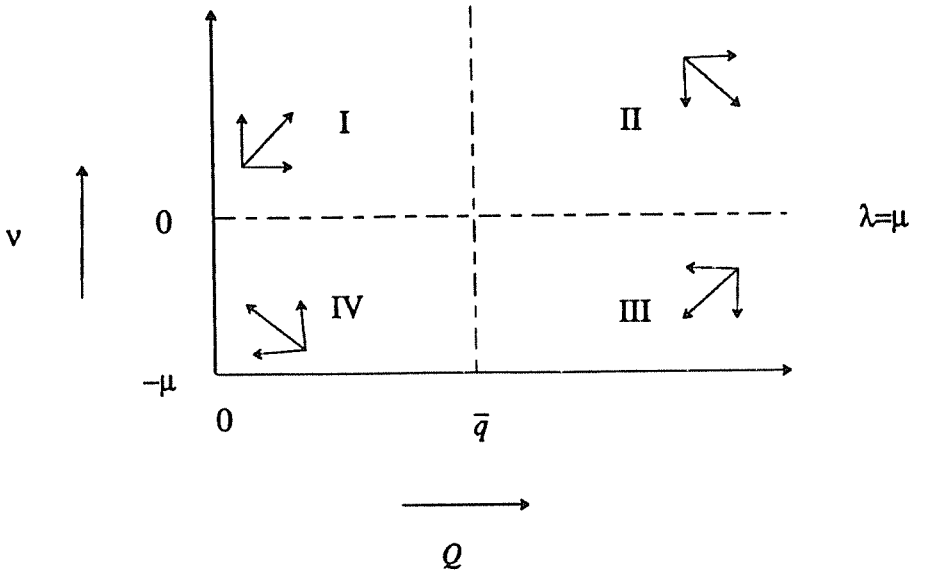

Figure 6.1: Characteristics and their directions.

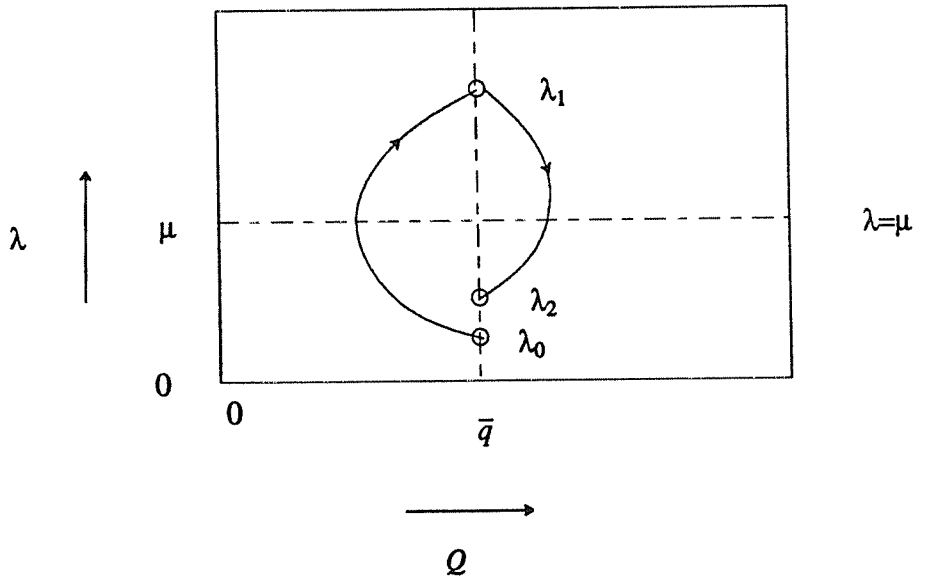

Figure 6.2: Converging spiral.

Therefore, from the above argument, it is clear that the trajectory would either be a *cycle* or a *spiral;* the latter could be one that *converges inwards* or *diverges outward.* Further, a convergent spiral could home in to either a *limit point* or a *limit cycle.* Theorem 6.1 below says that the path of any particle obeying Equations 6.2 and 6.14 (ignoring the σ^2 term) is a *convergent cycle* with the *limit point* $Q = \bar{q}$ and $\nu = 0$. Notice that this is exactly the desired point of operation of the adaptive algorithm.

Theorem 6.1:

If $\sigma^2 = 0$ in Equation 6.14, then Algorithm 6.2 converges in the limit. The *limit point* is $q = \bar{q}, \lambda = \mu$. This result is due to Prof. John Strikwerda.

Proof:

We have

$$\frac{dq}{dt} = \lambda - \mu \tag{6.15}$$

and

$$\frac{d\lambda}{dt} = g(\lambda, q) = \begin{cases} +C_0, & \text{if } q \leq \bar{q}, \\ -C_1\lambda, & \text{if } q > \bar{q}. \end{cases} \tag{6.16}$$

Since μ, the average service rate, is not changing with time,

$$\begin{aligned} \frac{d^2q}{dt^2} = g(\lambda, q) &= g(\mu + dq/dt, q) \\ &= \begin{cases} +C_0, & \text{if } q \leq \bar{q}, \\ -C_1\lambda, & \text{if } q > \bar{q}. \end{cases} \end{aligned} \tag{6.17}$$

Now, suppose that at time $t = 0$, λ is some value λ_0 which is less than μ and q is $\bar{q}$ (see Figure 6.2). From Equation 6.17, we have

$$\frac{d^2q}{dt^2} = C_0.$$

Its solution is

$$q = \frac{1}{2}C_0t^2 + (\lambda_0 - \mu)t + \bar{q} \tag{6.18}$$

After a certain time, say t_1, the characteristic hits $q = \bar{q}$ line again. Let λ be λ_1 now. For the moment, let us assume that the characteristic did not hit the $q = 0$ boundary, so that Equation 6.18 is valid all the way upto $t = t_1$.

The two roots of Equation 6.18 with $q = \bar{q}$ are $t = 0$ and $t = 2(\mu - \lambda_0)/C_0$. The first one corresponds to the initial point. Therefore,

$$t_1 = \frac{2(\mu - \lambda_0)}{C_0} \tag{6.19}$$

Also, since $\lambda = \mu + dq/dt$, we have, from Equation 6.18 and 6.19,

$$\begin{aligned} \lambda_1 &= \mu + C_0 t_1 + (\lambda_0 - \mu) \\ &= 2\mu - \lambda_0 \end{aligned} \tag{6.20}$$

Notice that $\lambda_1 - \mu$ is equal to $\mu - \lambda_0$ which says that the *overshoot* above μ is *exactly equal to* $\mu - \lambda_0$, *irrespective of the value of* C_0. This is therefore an inherent property of the *linear* increase component of Algorithm 6.2.

Let us next evaluate the characteristic when q is greater than $\bar{q}$. We have

$$\frac{dq}{dt} = \lambda - \mu \tag{6.21}$$

and

$$\begin{aligned} \frac{d^2q}{dt^2} &= g(\lambda, \mu) = -C_1(\mu + dq/dt) \\ \Longrightarrow \frac{d^2q}{dt^2} &+ C_1 \frac{dq}{dt} + C_1\mu = 0 \end{aligned} \tag{6.22}$$

Since at $t = t_1, q = \bar{q}$ and $dq/dt = \lambda_1 - \mu$, its solution is

$$q = -\mu(t - t_1) + \frac{\lambda_1}{C_1}\left(1 - e^{-C1(t-t_1)}\right) + \bar{q} \tag{6.23}$$

Let the characteristic again hit the $q = \bar{q}$ line at some later time t_2 and let λ now be λ_2. Then from Equation 6.23, we have at time t_2,

$$-\mu(t_2 - t_1) + \frac{\lambda_1}{C_1}\left(1 - e^{-C1(t_2-t_1)}\right) = 0$$

Putting $\alpha = C_1(t_2 - t_1)$, we get

$$\mu\alpha = \lambda_1(1 - e^{-\alpha}). \tag{6.24}$$

Since dq/dt is equal to $\lambda - \mu$, λ_2 can be obtained by differentiating Equation 6.23. We get

$$\begin{aligned} \lambda_2 &= \lambda_1 e^{-C1(t_2-t_1)} \\ &= \lambda_1 e^{-\alpha} \end{aligned} \tag{6.25}$$

Substituting the value of λ_1 from Equation 6.20, we have

$$\lambda_2 = (2\mu - \lambda_0)e^{-\alpha} \tag{6.26}$$

Therefore

$$\frac{\lambda_2}{\lambda_0} = (2\frac{\mu}{\lambda_0} - 1)e^{-\alpha} \tag{6.27}$$

The question then is whether λ_2/λ_0 is greater than 1, less than 1 or equal to 1. From Figure 6.2, we see that *if* λ_2/λ_0 were greater than 1, we would have a converging spiral. We verify next that this is indeed the case.

Let $\gamma = \mu/\lambda_1$ in Equation 6.24. Then, using Equation 6.20, we have

$$\gamma = \frac{\mu}{\lambda_1} = \frac{\mu}{2\mu - \lambda_0} = \frac{1}{2 - \lambda_0/\mu}$$

$$\Rightarrow 2 - \frac{\lambda_0}{\mu} = \gamma^{-1} \tag{6.28}$$

Substituting into Equation 6.27, we get

$$\begin{aligned} \frac{\lambda_2}{\lambda_0} &= \left[\frac{2}{2 - \gamma^{-1}} - 1\right] e^{-\alpha} \\ &= \left[\frac{2\gamma}{2\gamma - 1} - 1\right] e^{-\alpha} \\ &= \left[\frac{1}{2\gamma - 1}\right] e^{-\alpha} \end{aligned} \tag{6.29}$$

From Equation 6.24, γ is given by

$$\gamma = \frac{1 - e^{-\alpha}}{\alpha}$$

Therefore,

$$2\gamma - 1 = \frac{2(1 - e^{-\alpha})}{\alpha} - 1 = \frac{2 - \alpha - 2\alpha e^{-\alpha}}{\alpha} \tag{6.30}$$

and from 6.29 and 6.30,

$$\frac{\lambda_2}{\lambda_0} = \frac{\alpha e^{-\alpha}}{2 - \alpha - 2\alpha e^{-\alpha}} \tag{6.31}$$

Let us next define a function, $h(\alpha)$, such that

$$h(\alpha) = (2 - \alpha - 2e^{-\alpha}) - \alpha e^{-\alpha} \tag{6.32}$$

If $h(\alpha)$ is less than 0, then from Equation 6.31, λ_2/λ_0 is greater than 1. Notice that $h(0)$ is 0 and

$$h'(\alpha) = -1 + e^{-\alpha} + \alpha e^{-\alpha}.$$

So,

$$h'(0) = 0.$$

Differentiating once again,

$$h''(\alpha) = -\alpha e^{-\alpha} < 0 \qquad \text{for} \qquad \alpha > 0$$

Therefore,

$$h'(\alpha) = \int_0^\alpha h''(\alpha) < 0$$

Similarly,

$$h(\alpha) = \int_0^\alpha h'(\alpha) < 0 \tag{6.33}$$

From Equations 6.31, 6.32 and 6.33, we have

$$\frac{\lambda_2}{\lambda_0} > 1 \tag{6.34}$$

which implies that *the spiral is convergent.*

So far, we have assumed that the characteristic starting at $(\bar{q}, \lambda_0)$ never hits the $q = 0$ boundary. In this case, we have established that we have a convergent spiral. To complete the proof, let us next consider the case when a characteristic hits the left boundary, $q = 0$.

Notice that this characteristic cannot hit the boundary for $\lambda > \mu$, because the q-drift which is positive for $\lambda > \mu$, will pull it to the right. Therefore, if it hits the $q = 0$ boundary then $\lambda \leq \mu$. Suppose that for some initial $(\bar{q}, \hat{\lambda_0})$, the characteristic barely *touches* the boundary. This point is $(q = 0, \lambda = \mu)$, as shown by arc 'a' in Figure 6.3. Since Equations 6.18, 6.19 and 6.20 hold for this characteristic, it will converge by the earlier argument. Any point corresponding to $\lambda_0 < \hat{\lambda_0}$ first hits the $q = 0$ boundary (as shown by arc e), then goes vertically up until $\lambda = \mu$, (arc f), and then follows the characteristic corresponding to $\hat{\lambda_0}$, (arcs b, c, d). This too, therefore, converges. The pde 6.14 is however, not quite valid in this range.

This completes the proof of Theorem 6.1. ∎

Corollary 1: If both the increase and the decrease components are *linear,* then the system will never converge.

Proof:

We saw from Equation 6.20 that the amount of overshoot exactly equals the amount of undershoot during the linear increase phase *irrespective of the value of* C_0. The same is true in the reverse direction for a linear decrease algorithm. Hence, the system moves in a non-convergent cycle. ∎

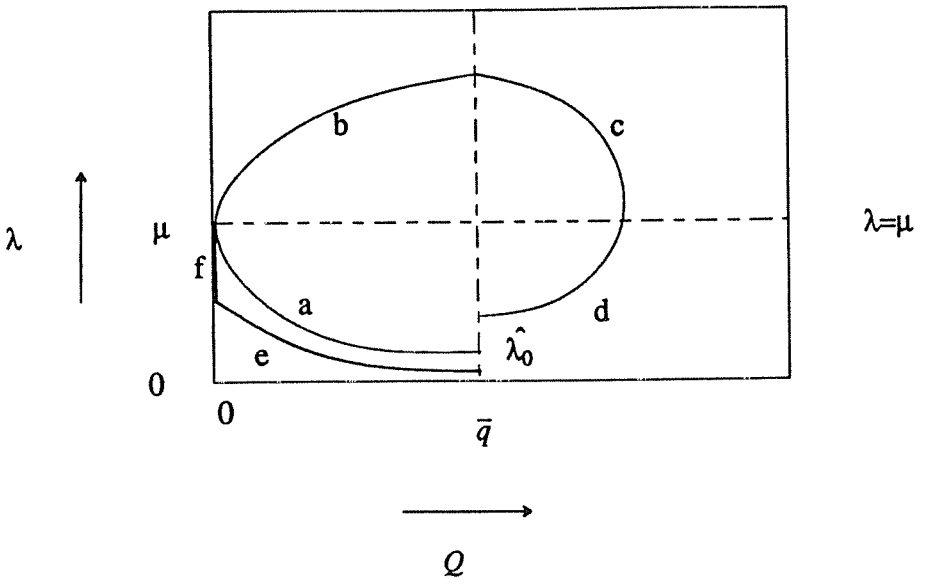

Figure 6.3: Converging spiral when characteristics touch the q=0 boundary.

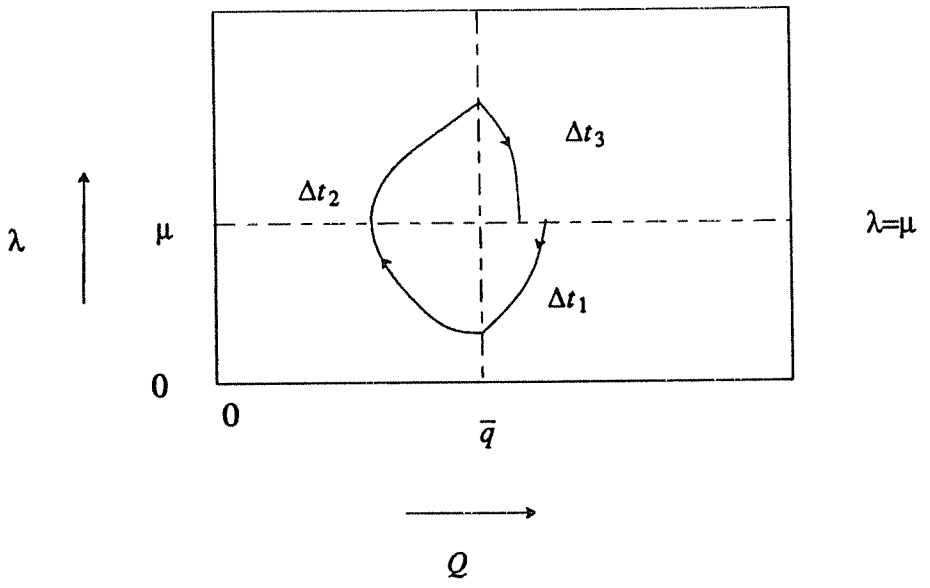

Figure 6.4: Meaning of $\Delta t_1, \Delta t_2$ and Δt_3.

6.5. Multiple Sources

We have assumed so far that there is only a single source transmitting through a particular node. We next investigate the properties of the system with multiple sources. Specifically, we are interested in the *convergence* and *fairness* properties when multiple sources compete for a resource. There are two 'feedback schemes' that we consider; one where all the sources receive the (same) cumulative queue length information [RaJa 88, Jac 88] and another, where each source receives its own queue length information only.[4] In the latter case, fairness is guaranteed by the scheduler; the analysis of the previous sections then apply directly to each source; if there are n sources, we change μ to μ/n and apply Equations 6.2 and 6.12. The conclusion is that the system is both convergent and fair.

Next, let us consider the case when all sources receive the common queue length information. All of them adjust their rates according to Algorithm 6.2. If there are n sources, let $(\lambda_1(t), \lambda_2(t), \cdots, \lambda_n(t))$ denote their transmission rates at time t. Let $\lambda(t) = \sum_{i=1}^{n} \lambda_i(t)$ be the cumulative transmission rate and let $Q(t)$ be the cumulative queue length at time t. Then

$$\frac{d}{dt}\lambda(t) = \sum_{i=0}^{n} \frac{d}{dt}\lambda_i(t) = \begin{cases} +nC_0, & \text{if } Q(t) \leq \bar{q}, \\ -C_1 \sum_i \lambda_i(t) = -C_1\lambda(t), & \text{if } Q(t) > \bar{q}. \end{cases} \quad (6.35)$$

This is the equivalent version of Equation 6.2 for multiple sources. Equations 6.12 and 6.35 completely specify the behavior of the system. From Theorem 6.1, this system of multiple sources *converges*. Notice that the increase rate is proportional to n, but the decrease rate is unchanged. Therefore, the length of the spiral trajectory (the path to convergence) is the same, but the time to traverse it is shortened (see Equations 6.18 and 6.19).

We next investigate if Algorithm 6.35 is *fair*. If it is, then the $\lambda_i's$ must be equal to each other in the limit.

Theorem 6.2:

---

[4] possible with a Fair-Queue-like scheduling algorithm at the resource.

Algorithm 6.35 is *fair*.

Proof:

The proof is due to Prof. John Strikwerda.

For the purposes of this proof, let us assume that the different sources have the same increase parameter C_0 but different decrease parameters $c_1, c_2, \cdots, c_n$ instead of a single C_1.[5] Let $\lambda_1, \lambda_2, \cdots, \lambda_n$ denote their transmission rates in the limit (notice that convergence is guaranteed by Theorem 6.1). Then

$$\lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_n = \mu \tag{6.36}$$

Suppose $\lambda_1^o, \lambda_2^o, \cdots, \lambda_n^o$ are the transmission rates at some time such that

$$\lambda_1^o + \lambda_2^o + \cdots + \lambda_n^o = \mu$$

but let q be greater than $\bar{q}$ (see Figure 6.4). Let Δt_1, Δt_2 and Δt_3 be as shown in the figure. These are three disjoint segments of the time to complete one complete cycle.[6] Let $\lambda_1^1, \lambda_2^1, \cdots, \lambda_n^1$ be the new values of the $\lambda_i's$ at the end of the cycle. Then, the equation for λ_1^1 is given by

$$\lambda_1^1 = (\lambda_1^o e^{-c_1 \Delta t_1} + C_0 \Delta t_2) e^{-c_1 \Delta t_3}$$

Other λ_i^1's are similar. We then get,

$$\frac{\lambda_1^1 - \lambda_1^o}{\Delta t_2} = \frac{\lambda_1^o \left(e^{-c_1(\Delta t_1 + \Delta t_3)} - 1\right)}{\Delta t_2} + C_0 e^{-c_1 \Delta t_3} \tag{6.37}$$

Let $\gamma = (\Delta t_1 + \Delta t_3)/\Delta t_2$. Then passing Equation 6.37 to the limit as $\Delta t_2 \to 0$ which will occur as the processes tend to equillibrium, we get

$$\frac{d\lambda_1}{dt_2} = -c_1 \gamma \lambda_1 + C_0 \tag{6.38}$$

---

[5] this is therefore, a more general proof.

[6] i.e., when the process hits $\lambda = \mu$ and $q > \bar{q}$ again.

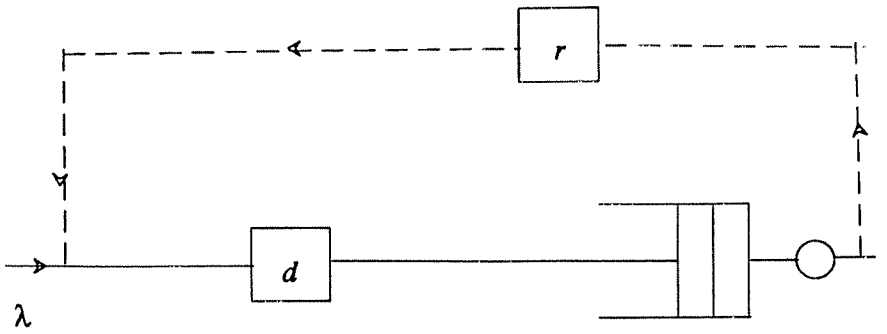

Figure 6.5: Delayed feedback.

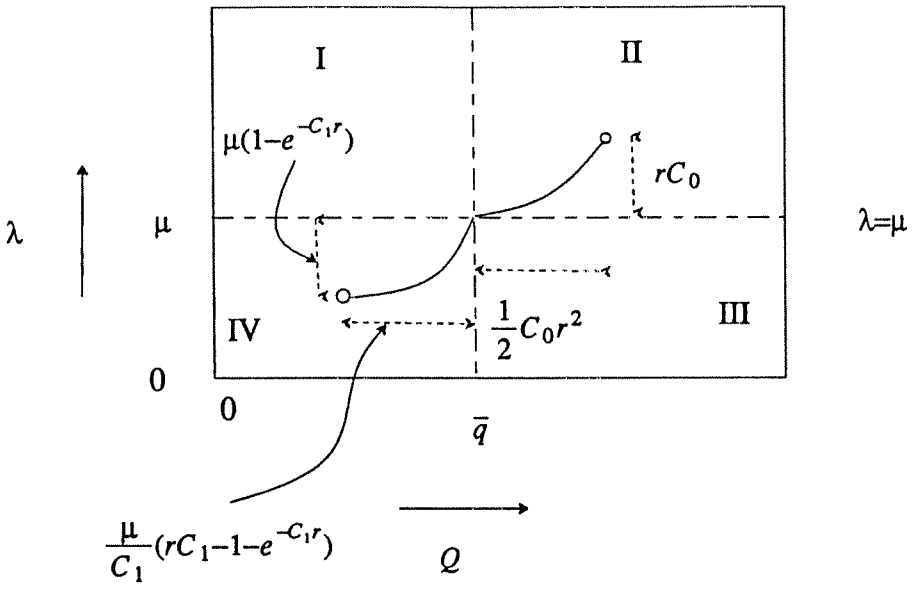

Figure 6.6: Consequence of delayed feedback.

Similarly,

$$\frac{d\lambda_i}{dt_2} = -c_i\gamma\lambda_i + C_0 \tag{6.39}$$

In the limit, when convergence occurs,

$$\frac{d\lambda_i}{dt_2} = 0,$$

so

$$\lambda_i = \frac{C_0}{\gamma c_i} \tag{6.40}$$

Since, $\sum_i \lambda_i = \mu$, we have

$$\frac{C_0}{\gamma} = \frac{\mu}{\sum_i 1/c_i}$$

Therefore

$$\lambda_i = \frac{\mu}{c_i \sum_{j=1}^{n} 1/c_j} \tag{6.41}$$

Thus, if the $c_i's$ are equal, then $\lambda_i = \mu/n$, which implies complete fairness. ∎

6.6. Effect of feedback delay

We next investigate the effect of feedback delay on the control algorithm. Figure 6.5 shows the mechanics of the system; r is the delay in obtaining the feedback information from the queue to the control point; d is the inertia in the forward direction in that it takes the control algorithm this much time to take effect after λ is changed. Let us, for the moment, assume that d is 0.

The control algorithm can now be precisely stated as:

$$\frac{d}{dt}\lambda(t) = \begin{cases} +C_0, & \text{if } Q(t-r) \leq \bar{q}, \\ -C_1\lambda(t), & \text{if } Q(t-r) > \bar{q} \end{cases} \tag{6.42}$$

It turns out that this algorithm does not converge. To see this, suppose that at time t_0, the process is at the target equilibrium point $Q(t_0) = \bar{q}$

and $\lambda(t_0) = \mu$. We shall show that it cannot remain here for any significant amount of time.

We need to consider two cases. First, let us say that the process arrived at this point from the left, i.e., $Q(t_0 - r) < \bar{q}$. Then

$$d\lambda(t)/dt = C_0, \qquad t \in (t_0, t_0 + r) \tag{6.43}$$

Therefore

$$\lambda(t_0 + r) = \lambda(t_0) + rC_0 = \mu + rC_0 > \mu \tag{6.44}$$

and

$$Q(t_0 + r) = \bar{q} + \frac{1}{2}C_0 r^2 > \bar{q} \tag{6.45}$$

Figure 6.6 shows this pictorially (see Quadrant II). The process overshoots the equilibrium point because $r > 0$.

Next, let us consider the case when the process arrives at $(\bar{q}, \mu)$ from the right, i.e., $Q(t_0 - r) > \bar{q}$. Then

$$d\lambda(t)/dt = -C_1\lambda(t), \qquad t \in (t_0, t_0 + r) \tag{6.46}$$

Therefore

$$\lambda(t_0 + r) = \lambda(t_0)e^{-C_1 r} = \mu e^{-C_1 r} < \mu \tag{6.47}$$

and

$$Q(t_0 + r) = \bar{q} - \frac{\mu}{C_1}\left(rC_1 - 1 + e^{-C_1 r}\right) < \bar{q} \tag{6.48}$$

Figure 6.6 shows this case too (Quadrant IV). The process, here, undershoots the equilibrium.

Notice that the overshoot and the undershoot are going to be larger than what is shown above because when $Q(t_0) = \bar{q}$, the value of λ will either be greater than μ or less than μ (depending on whether the process came from left or right respectively). Clearly the system cannot stabilize at $(\bar{q}, \mu)$. Further, at any other point in the $q - \lambda$ space, the process is forced to be in motion. Therefore the system oscillates. The diameter of the oscillatory cycle increases with the delay, r. If different sources experience

different delays, they have different oscillatory cycles. This could lead to unfairness in resource usage.

Equations 6.44, 6.45, 6.47 and 6.48 point to an important difficulty with choosing parameters C_0 and C_1. The oscillations are larger with higher values of of these parameters. Thus, while larger values of C_0 and C_1 help to converge faster in the absence of delay (see Equation 6.18 for example), they cause larger oscillations in the presence of delay.

Next, let us consider the effect of the inertia d. We still have

$$\frac{d}{dt}\lambda(t) = \begin{cases} +C_0, & \text{if } Q(t-r) \leq \bar{q}, \\ -C_1\lambda(t), & \text{if } Q(t-r) > \bar{q} \end{cases} \tag{6.49}$$

However, d^2Q/dt^2 is now given by

$$\frac{d^2}{dt^2}Q(t) = \frac{d}{dt}\lambda(t-d) = \begin{cases} +C_0, & \text{if } Q(t-r-d) \leq \bar{q}, \\ -C_1\lambda(t), & \text{if } Q(t-r-d) > \bar{q} \end{cases} \tag{6.50}$$

i.e., the queue length now lags $r+d$ time units while λ still lags r time units. The oscillatory effect is now more severe, but qualitatively similar to the previous case, i.e., larger values of C_0 and C_1 cause larger oscillations.

6.7. Summary and conclusions

We presented an approximate analysis of a queue with dynamically changing input rates based on implicit or explicit feedback. This was motivated by recent proposals for adaptive congestion control algorithms [RaJa 88, 90, Jac 88], where the sender's window size at the transport level was adjusted based on perceived congestion level of a bottleneck node. We developed an analysis *methodology* for a simplified system; yet it was powerful enough to answer the important questions regarding stability, convergence (or oscillations), fairness and the significant effect that delayed feedback plays on performance. Specifically, we found that, in the absence of feedback delay, the linear increase/exponential decrease algorithm of Jacobson and

Ramakrishnan-Jain [Jac 88, RaJa 88] was *provably* stable and fair. Delayed feedback, on the other hand, introduced oscillations for *every* individual user as well as unfairness across those competing for the same resource. While simulation studies of Zhang [Zha 89] had observed the oscillations in the cumulative queue length at the bottleneck and measurements by Jacobson [Jac 88] had revealed some of the unfairness properties, the *reasons* for these had not been properly understood. We identified *quantitatively* the real *cause* for the these effects.

We found that introduction of feedback delay however added *oscillations* which settle down to a *limit cycle,* i.e., a cyclic pattern that was constant in the limit. This cyclic pattern agreed with simulation results by Zhang [Zha 89]. The proof of the existence of a limit cycle, we believe, is a new result. The diameter of the limit cycle (or equivalently the magnitude of the oscillations) was seen to be sensitive to the parameters C_0, C_1 and the feedback delay. For instance, for a fixed C_0 and feedback delay, a larger C_1 *increased* this diameter. So, while in the absence of feedback delay, a larger C_1 boosted the speed of convergence, in the presence of delay, it caused wilder oscillations. The size of the oscillations also increased with C_0 and feedback delay.

Our model is fairly general and is applicable to evaluate the performance of a wide range of feedback control schemes. It is an extension of the classical Fokker-Planck equation. Therefore, it addresses traffic variability (to some extent) that fluid approximation techniques do not.

Chapter 7

Future Work

7.1. Introduction

In this chapter, we outline our plans for future research. We wish to explore two major avenues in the near future.

One avenue is congestion control in high speed wide area networks. The other is extending the Fokker-Planck analysis of feedback control algorithms with delay in the feedback path.

7.2. Congestion control in high speed, wide area networks

Introduction of optical fibers is pushing transmission speeds to the gigabit range. While this offers new dimensions to networking, the challenge we face is to pass these hardware speeds to the applications that will use it. A stiff performance hurdle is the high bandwidth-delay product of these networks when propagation delay is large. The round trip propagation delay across continental USA in fiber is approximately 46 msec. At gigabit speeds, one could dump $10^9 * 46 * 10^{-3}$ bits (= 5.75 MBytes) of data into the network before hearing from the receiver. Consider now, a reactive congestion control scheme that uses (implicit or explicit) feedback information from the network to adjust the input transmission rate. The feedback information is potentially old in this environment, relative to the duration of *short-term* fluctuations in queue length caused by bursty traffic. (Feedback may still

be used to track long term fluctuations in traffic). To deal with short term bursts, numerous 'open-loop' strategies have recently been proposed [Zha 89, SLCG 89, Tur 86, BCS 90, Gol 90]. We propose a new strategy which we believe will perform better. A careful comparative study needs to be performed however, to get concrete performance answers.

Before we delve into the details of open-loop control, we take a closer look at the problem of packet loss once more. We shall then address possible solution methods including those that have been proposed recently by others.

The Problem

Suppose that a message of size N packets is being transmitted over a network. Let p_j be the probability of packet loss for packet j with a particular flow control protocol and a particular retransmission strategy. Let τ be the average cost of an error. If the p_j were statistically independent and identical, we know from Chapter 3 that the expected time to transmit the N-packet message will be

$$E[T_N] = E[T_{N,NoErrs}] + N\frac{p_j}{1-p_j}\tau$$

If errors were correlated however, we get a weaker relation. Let $p = \sup_j p_j$. Then

$$E[T_N] \leq E[T_{N,NoErrs}] + N\frac{p}{1-p}\tau$$

$$\approx E[T_{N,NoErrs}] + Np\tau$$

because $p << 1$. Let us define loss of efficiency, η, as

$$\eta = \frac{E[T_N] - E[T_{N,NoErrs}]}{E[T_{N,NoErrs}]}$$

Then, using the relation $E[T_{N,NoErrs}] = Nt_1 + t_{end}$, ($cf.$ Chapter 3), we have

$$\eta \leq Np\tau\frac{1}{Nt_1 + t_{end}}$$

$$\leq Np\tau\frac{1}{Nt_1}$$

$$= p\left(\frac{\tau}{t_1}\right) \tag{7.1}$$

This relation shows quantitatively, how efficiency scales (or decays) with transmission speed. For the go-back-n protocol, τ is at least equal to the roundtrip propagation delay. Increase in transmission speeds to the gigabit range would decrease t_1 10 to 100 fold, so p would have to decrease by the same amount to keep efficiency comparable.[1] Equation 7.1 is therefore a 'rule-of-thumb' design equation.

For selective repeat, the value of τ depends on N and flow control. If the pipe were kept full for significant amounts of time, the cost due to the loss of a packet would be low. However, this would be difficult in high speed environments on two counts. First, most message bursts would not be large enough to keep the pipe full. Second, high bandwidth-delay product would require large receive buffers for caching out of order data.

The solution that we seek is to reduce packet loss without introducing negative side effects like increased queueing delays or slowing down transmissions unnecessarily. To this end, let us first consider some of the proposals that have been made recently.

Related Work

The *Leaky Bucket Protocol* [Tur 86, SLCG 89] provides a bucket of finite size at the input. The bucket is supplied with tokens which arrive at some specified rate γ. Tokens which arrive when the bucket is full are discarded. When packets from a user arrive, they each grab a token (if available) and

---

[1] The $\leq$ sign in Equation 7.1 does not make this argument any weaker. It can easily be checked that $\eta \approx \frac{1}{N}\sum_j p_j\tau/t_1$. Consider next a loss-curve for p_j vs j. A higher value of $p = \sup_j p_j$ is likely to increase all the p_j's because they are congestion-related.

get in the network. If the bucket is empty, the packet waits for a new token to arrive and only then does it have permission to go in.

The *Generalized Leaky Bucket Protocol* [BCS 90] provides two buckets at the input instead of one. These are called the Green Bucket and the Red Bucket and are fed at rates γ_g and γ_r respectively. An incoming packet grabs a token from the green bucket if the latter is non-empty and enters the network. Otherwise, it tries the red bucket, failing which it waits. Packets with green tokens are given a higher priority at intermediate queues. The idea of Generalized Leaky Bucket is to allow users to exceed the one-level burst size, but at their own risk. The important problem is to determine what γ_g and γ_r should be, to ensure low loss rates and yet higher throughput. No such study has yet been reported.

The *Virtual Clock Protocol* [Zha 89] takes the approach of *allocating* part of intermediate resources to individual users based on their *average* demands. It is a reservation based scheme. Let us suppose that a user (or a 'flow' as Zhang calls it) i is allocated a rate λ_i. Each node associated with this user maintains a *virtual clock* v_i which determines user i's priority with respect to other competing users. Initially, v_i is set to the real clock. When a packet belonging to user i arrives, v_i is incremented to $v_i + 1/\lambda_i$. The packet that is scheduled next for dispatching belongs to the user with the minimum virtual clock.

The Leaky Bucket Protocol allows bursts of up to size K at the entrance to the network. Intermediate nodes could however see larger burst sizes due to cumulative effect of multiple bursts from different users. Thus this protocol offers only limited protection from buffer overruns and large queueing delays. The Virtual Clock Protocol on the other hand maintains traffic smoothness across tandem links if the traffic is smooth at the entry point. However, in so doing, it forces bursty sources to transmit at their average rates or face the penalty of large delays and/or packet loss.

Stop-and-Go Queueing [Gol 90] ensures that the smoothness of traffic at the input is maintained across multiple hops. While Golestani proposed it for real-time traffic, it is relevant to data traffic as well. The basic idea of

Stop-and-Go Queueing is to slow down traffic on a per user basis at every intermediate node so that packets which were separated out at the input do not arrive in close succession at some node in the network. This latter phenomenon could occur if successive packets of a user get queued at some intermediate node. Golestani shows that Stop-and-Go Queueing can ensure lossless transmission with a bounded number of buffers. Unfortunately this is at the expense of throttling traffic, possibly unnecessarily.

Notice that, unlike Virtual Clock, Stop-and-Go Queueing is not a priority scheduling algorithm. In Virtual Clock, the average specified rates of individual users are used to determine who goes next. If a user transmits faster than its specified average rate, it still gets scheduled if the resource is free. This is not possible with Stop-and-Go Queueing. On the other hand, Stop-and-Go Queueing does allow users more flexibility with respect to defining their smoothness (the average rate over a time period T).

New Proposal

We propose the *Gate* Protocol which combines the ideas of Leaky Bucket and Virtual Clock to provide good service to *packet trains* [RoJa 86, SoLa 88] across tandem links. In the packet train model for data traffic, a user is assumed to be in one of two states. In one state it transmits data with some rate λ. In the other, it remains idle. Let us suppose that a user i specifies its transmission rate λ_i, and its burst characteristics $t_{1,i}$ and $t_{2,i}$ as shown in Figure 7.1. λ_i corresponds to the speed that user i wishes to transmit at. It may be the maximum speed at which it can transmit or the speed decreed by flow control. We believe that users would be able to specify their packet-train characteristics with the help of statistics collectors. Further investigation is required to determine the accuracy of these predictions.

The Gate Protocol will work as follows. Let $N_i = \lambda_i t_{1,i}/\text{packet size}$. N_i is the number of packets that user i would transmit in time $t_{1,i}$. Associate a virtual clock v_i and a bucket B_i at every node in the path of user i. Initialize

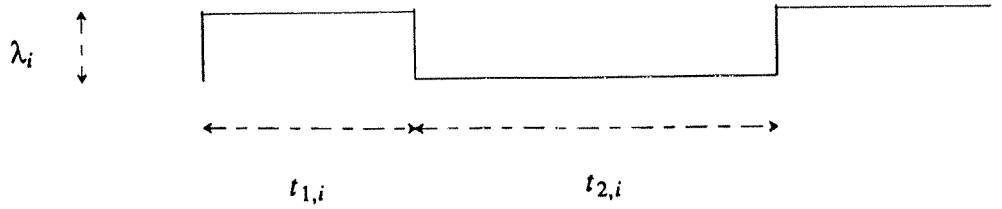

Figure 7.1. User specifications for Gate.

B_i to N_i and v_i to real clock.

When a packet from user i arrives at the node, execute the following algorithm.

---

priority[packet] := v_i
If ($B_i > 0$) then
  $v_i := v_i + 1/\lambda_i$
  $B_i := B_i - 1$
end
else
  $v_i := v_i + t_{2,i}$
  $B_i := N_i$
end
{ schedule packet with the minimum priority }

If a user behaves according to its own specifications, this scheduling algorithm can guarantee it good service in that its packets will get dispatched without excessive delay. Also, packet loss can be avoided altogether. However, if a user exceeds its specified burst size or transmits faster than agreed upon, then as in Virtual Clock, its priority is reduced and its own performance is affected.

Notice that Gate is identical to Virtual Clock if $t_{2,i} = 0$ (In this case, N_i could be any positive integer). Thus Gate allows users to specify only their average rates of transmission if they choose to. However, if a user provides more information to the network, it could get better service than either Virtual Clock or Leaky Bucket. Also, unlike Stop-and-Go, it does not throttle packets when a resource is idle. Therefore, it will deliver higher throughput and lower packet delays than Stop-and-Go. Gate and Virtual

Clock also provide protection from misbehaving users. Stop-and-Go does not.

Outline of Performance Study

We plan to compare the performance of these protocols using detailed simulation. The workload will be the number of users and their burst characteristics. The output of interest would be effective throughput, loss and delay as a function of offered load.

A subsequent study would be to determine analytically the tradeoffs involved in exceeding the specified rates as a function of congestion level. Design of *optimistic* versions of the protocols at the input would benefit from this information. One advantage in this reservation-based environment is that the nodes are aware of rough user demands. If they could compute levels of loss probabilities as a function of user demand, they could pass the information to the users. Since users are likely to persist longer than just a round-trip delay, the feedback information may be exploited to get better performance. Users could decide if they want to exceed their rates because they are aware of the costs associated with it.

Once the loss probabilities are determined, we would compare 'Optimistic' Gate with Generalized Leaky Bucket and 'Optimistic' Virtual Clock using simulation.

7.3. Fokker-Planck analysis of feedback control with delay

We would also explore ways to extend the Fokker-Planck analysis of Chapter 6 with a delayed feedback. At this point, the problem looks quite formidable and we do not know how we would go about solving it.

Bibliography

[AnPr 86]: Anagnostou, M.E. and E. N. Protonotarios, "Performance analysis of the selective repeat ARQ protocol," *IEEE Trans. Commun.*, vol. COM-34, pp. 127-135, Feb. 1986.

[BCS 90]: Bala, K., I. Cidon and K. Sohraby, "Congestion Control in High Speed Packet Switched Networks," *Proc. of the IEEE Infocom*, pp 520-526, June 1990.

[Bir 81]: Birrel, N.D., "Pre-emptive retransmission for communication over noisy channels," *IEE Proc., Part F,* vol. 128, pp 393-400, Nov 1981.

[BPS 88]: Bolot, J.C., B.D. Plateau and A. U. Shankar, "Performance analysis of transport protocols over congestive channels," Tech Report UMIACS-TR-88-22.1 CS-TR-2004.1, Department of Computer Science, University of Maryland, Revised August 1988.

[BrMo 86]: Bruneel, H. and M. Moeneclaey, "On the throughput analysis of some continuous ARQ strategies with repeated transmissions," *IEEE Trans. Commun.*, vol. COM-34, pp. 244-249, March 1986.

[BRT 88]: Blake, J.T., A.L. Reibman and K.S. Trivedi, "Sensitivity Analysis of Reliability and Performability Measures for Multiprocessor Systems," *Proc. of the ACM Sigmetrics,* pp 177-186, May 1988.

[FKYN 78]: Fujiwara, C., M. Kasahara, K. Yamashita and T. Namekawa, "Evaluations of Error Control Techniques in Both Independent Error and Dependent-Error Channels," *IEEE Trans. Commun.,* vol. COM-26, pp 785-794, June 1978.

[Gav 68]: Gaver, D.P., Jr., "Diffusion Approximations and Models for Certain Congestion Problems," *Journal of Applied Probability,* pp 607-623, 1968.

[GeKl 80]: Gerla, M. and L. Kleinrock, "Flow Control: A Comparative Survey," *IEEE Trans. Commun.,* vol. COM-28, pp 553-574, April 1980.

[Gol 90]: Golestani, S. J., "Congestion-Free Transmission of Real-Time Traffic in Packet Networks," *Proc. of the IEEE Infocom,* pp 527-536, June 1990.

[Gus 87]: Gusella, R. , "The Analysis of Diskless Workstation Traffic on an Ethernet," Tech. Rep. UCB/CSD 87/379, University of California, Berkeley. November 1987.

[Jac 88]: Jacobson, V., "Congestion Avoidance and Control," *Proc. of the ACM Sigcomm,* pp 314-329, Aug. 1988.

[Jaf 81a]: Jaffe, J.M., "Bottleneck Flow Control," *IEEE Trans. Commun.,* vol. COM-29, pp 954-962, July 1981.

[Jaf 81b]: Jaffe, J.M., "Flow Control Power is Nondecentralizable," *IEEE Trans. Commun.,* vol. COM-29, pp 1301-1306, September 1981.

[Jai 86]: Jain, R., "A Timeout-Based Congestion Control Scheme for

Window Flow-Controlled Networks," *IEEE Journal on Selected Areas in Communications,* pp 1162-1167, 1986.

[Jai 90]: Jain, R,, "Congestion Control in Computer Networks: Issues and Trends," *IEEE Network Magazine,* pp 24-30, May 1990.

[JaRo 85]: Jain, R., and S. Routhier, "Packet Trains: Measurements and a New Model for Computer Network Traffic," Tech Report MIT/LCSTM 292, Department of Electrical Engg, MIT, Nov 1985.

[JCH 84]: Jain, R., D.M. Chiu and W.R. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems," DEC Tech Rep TR-301, 1984.

[KaCh 88]: Kanakia, H. and D.R. Cheriton, "The VMTP Adapter Board (NAB): High-Performance Network Communication for Multiprocessors ," *Proc. of the ACM Sigcomm,* pp 175-187, Aug. 1988.

[Kin 70]: Kingman, J.F.C., "Inequalities in the Theory of Queues," *Journal of the Royal Statistical Society, Series B,* vol. 32, pp 102-110, 1970.

[Kon 80]: Konheim, A.G., "A Queueing Analysis of Two ARQ Protocols," *IEEE Trans. Commun.,* vol. COM-28, pp 1004-1014, July 1980.

[Kle 75]: Kleinrock, L., *Queueing Systems Vol I: Theory,* John-Wiley and Sons, New York, 1975.

[Kle 76]: Kleinrock, L., *Queueing Systems Vol II: Computer Applications,* John-Wiley and Sons, New York, 1976.

[LaRe 79]: Lam, S.S. and M. Reiser, "Congestion Control of Store-and-Forward Networks by Input Buffer Limits — An Analysis," *IEEE Trans. Commun.,* vol. COM-27, pp 127-133, Jan. 1979.

[LiYu 80]: Lin, S., and P.S. Yu, "An Effective Control Scheme for Satellite Communications," *IEEE Trans. Commun.,* vol. COM-28, pp 395-401, March 1980.

[MBC 84]: Marsan, A., M.G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets," *ACM Trans. Computer Systems,* vol 2, pp 93-122, May 1984.

[MeSw 80]: Merlin, P.M. , and P.J. Schweitzer, "Deadlock Avoidance in Store-and-Forward Networks — I: Store-and-Forward Deadlock," *IEEE Trans. Commun.,* vol. COM-28, pp 345-354, March 1980.

[MiMi 90]: Mitra, D. and I. Mitrani, "A Go-Back-N Protocol Which is Efficient in High Speed Data Networks with Small Buffers," *Proc. 4th Int'l Conference on Data Commun. Systems and Their Performance,* Barcelona, June 1990.

[MLS 89]: Mukherjee, A., L.H. Landweber and J.C. Strikwerda, "Evaluation of Retransmission Strategies in a Local Area Network Environment," *Proc. of the ACM Sigmetrics/Performance,* pp 98-107, May 1989.

[Moe 86]: Moeneclaey, M., "Throughput optimization for a generalized stop-and-wait ARQ scheme," *IEEE Trans. Commun.,* vol. COM-34, pp. 205-207, Feb. 1986.

[Mor 78]: Morris, J.M., "On Another go-back-n ARQ Technique for

High Error Rate Conditions," *IEEE Trans. Commun.,* vol. COM-26, pp 187-189, Jan. 1978.

[Mor 88]: Morgan, S., "Window Flow Control on a Trunked Byte-Stream Virtual Circuit," *IEEE Trans. Commun.,* vol. 36, pp 816-825, July 1988.

[MQR 87]: Mohan, S. , J. Qian and N.L. Rao, "Efficient Point-To-Point and Point-To-Multipoint Selective-Repeat ARQ Schemes with Multiple Retransmissions : A Throughput Analysis," *Proc. of the ACM Sigcomm Workshop,* pp 49-57, Aug. 1987.

[New 68]: Newell, G.F., "Queues with Time-Dependent Arrival Rates I-III," *Journal of Applied Probability,* pp 436-451, 1968.

[New 71]: Newell, G.F., *Applications of Queueing Theory,* Chapman and Hall Ltd., London, 1971.

[Oht 89]: Ohta, M., "Engineering of Congestion Control in a Large Scale Network," *Proc. 4th Int'l Network Planning Symposium,* Palma de Mallorca, Spain, pp 301-305, Sept. 1989.

[RaJa 88]: Ramakrishnan, K.K., and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer," *Proc. of the ACM Sigcomm,* pp 303-313, Aug. 1988. Also, *ACM Trans. Computer Systems,* vol. 8, pp 158-181, May 1990.

[Rei 79]: Reiser, M., "A Queueing Network Analysis of Computer Communication Networks with Window Flow Control," *IEEE Trans. Commun.,* pp 1199-1209, 1979.

[SaSc 80]: Saad, S. and M. Schwartz, "Input Buffer Limiting Mechanisms for Congestion Control," ICC, June 1980.

[Sas 75]: Sastry, A.R.K., "Performance of Hybrid Error Control Schemes on Satellite Channels," *IEEE Trans. Commun.,* vol. COM-23, pp 689-694, July 1975.

[SLCG 89]: Sidi, M., W. Liu, I. Cidon and I. Gopal, "Congestion Control Through Input Rate Regulation," *Proc. IEEE Globecom,* 1989.

[SoLa 88]: Song, C. and L.H. Landweber, "Optimizing Bulk Data Transfer: A Packet Train Approach," *Proc. of the ACM Sigcomm,* pp 134-145, Aug. 1988.

[SSSGJ 88]: Sanghi, D., M.C.V. Subramaniam, A. U. Shankar, O. Gudmundsson and P. Jalote, "Instrumenting a TCP Implementation," Tech. Report CS-TR-2061, Dept. of Computer Science, University of Maryland, July 1988.

[Tan 81]: Tanenbaum, A.S., *Computer Networks,* Prentice-Hall Inc., Englewood Cliffs, NJ 07632, 1981.

[Tow 79]: Towsley, D., "The Stutter Go-Back-N ARQ Protocol," *IEEE Trans. Commun.,* vol. COM-27, pp 869-875, June 1979.

[ToWo 79]: Towsley, D. and J.K. Wolf, "On the Statistical Analysis of Queue Lengths and Waiting Times for Statistical Multiplexors with ARQ Retransmission Schemes," *IEEE Trans. Commun.,* vol COM-27, pp 693-702, April 1979.

[Tri 82]: Trivedi, K.S., *Probability and Statistics with Reliability, Queuing, and Computer Science Applications,* Prentice-Hall Inc.,

Englewood Cliffs, NJ 07632, 1982.

[Tur 86]: Turner, J., "New directions in Communications (or Which way to the Information Age?)," *IEEE Comm. Magazine,* vol. 24, Oct 1986.

[Zha 89]: Zhang, L., "A New Architecture for Packet Switching Network Protocols," Ph.D. thesis, MIT, Dept. of Electrical Engg. and Computer Science, Aug. 1989.

[Zwa 85]: Zwaenepoel, W., "Protocols for Large Data Transfers on Local Networks, *Proc. of the ACM Sigcomm,* pp 22-32, 1985.

Appendix 3.A

This section presents the analysis of the time to detect an error, as discussed in Section 3.3.1. Given M+1 packets, of which the *first* packet has failed, we are interested in the time the sender takes to detect the error. We are assuming that errors due to electrical noise are much lower than errors due to packet losses at the interface. In a single hop LAN, all packets are received in the order sent. Therefore the receiver can detect a dropped packet with sequence number s if it receives any packet with sequence number of $s+1$ or greater. It then NACKs sequence number s. If the NACK gets through successfully, the error is detected at the sender; otherwise, a NACK from a future packet is needed for error detection. Ultimately, if there are no packets left (i.e., all M packets or their NACKs failed), the sender times out after $T_{timeout}$ time units. So error detection at the sender is upper bounded by $T_{timeout}$.

Now let

$q_0 = (1 - p_0)$, the probability that a packet does not fail.

$u = 1 - {q_0}^2$, the probability that a packet exchange fails

Then,

$$Pr[i \text{ failures to detect}] = \begin{cases} (1-u)u^i, & \text{if } 0 \le i \le M-1, \\ u^M, & \text{if } i = M. \end{cases} \qquad (3.A.1)$$

Distribution of time to detect

In the following discussion, C and T are defined as in Section 3.3.1. T_n is the time to transmit a NACK packet, and Cn is the time to copy it from (to) the interface memory to (from) the host memory. We assume that $Cn \le T$.

Let

$T_{start} = C + T$

$$T_{end}{}^{(1)} = (C+T) + (C+Cn)$$
$$T_{end}{}^{(2)} = (C+T) + (Cn+T_n)$$
$$T_{end}{}^{(3)} = (C+Cn) + (Cn+T_n)$$

Then the time to detect the error after exactly i failures, T_i is

$$T_i = \begin{cases} T_{start} + T_{end}{}^{(1)} + (i+1)(C+T), & \text{if } 0 \le i \le M-3, \\ T_{start} + T_{end}{}^{(2)} + (i+1)(C+T), & \text{if } i = M-2, \\ T_{start} + T_{end}{}^{(3)} + (i+1)(C+T), & \text{if } i = M-1, \\ T_{timeout}, & \text{if } i = M. \end{cases} \tag{3.A.2}$$

The interested reader may verify these equations by drawing the appropriate timing diagrams. The mean time to detect the error given M+1 packets is now easily obtained from Equations 3.A.1 and 3.A.2:

$$\begin{aligned} \tau(M+1) = T_{start} \sum_{i=0}^{M-1} (1-u)u^i + T_{end}{}^{(1)} \sum_{i=0}^{M-3} (1-u)u^i \\ + T_{end}{}^{(2)}(1-u)u^{M-2} + T_{end}{}^{(3)}(1-u)u^{M-1} \\ + (C+T) \sum_{i=0}^{M-1} (1-u)u^i(i+1) + T_{timeout}u^M \end{aligned}$$

which simplifies to

$$\begin{aligned} \tau(M+1) &= T_{start}(1-u^M) + T_{end}{}^{(1)}(1-u^{M-2}) \\ &\quad + T_{end}{}^{(2)}(1-u)u^{M-2} + T_{end}{}^{(3)}(1-u)u^{M-1} \\ &\quad + (C+T)\left[\frac{1-(M+1)u^M + Mu^{M+1}}{1-u}\right] + T_{timeout}u^M \qquad (3.A.3) \\ &= T_{start} + T_{end}{}^{(1)} + C + T + O\left(M(2p_0)^M\right) \qquad (3.A.4) \end{aligned}$$

This gives the mean time to detect an error if the receiver NACKs an erroneous packet. The time to detect an error turns out to be almost a constant. Low packet loss rates makes it extremely unlikely that consecutive errors will occur. Most of the time, a NACK will arrive almost immediately.

Thus error *recovery* is quick if M is large. It is almost independent of the timeout $T_{timeout}$, because of the *feedback control* provided by the NACK. Blast protocols with NACK and complete retransmission on error have also been shown to be independent of $T_{timeout}$ [Zwa 85]. Tuning $T_{timeout}$ to a very low value to reduce the time to recovery is another possible solution, but it is a *feedforward control* and can lead to needless retransmissions.

Appendix 3.B

In this appendix, we compute the covariance of the random variables X and Y of Section 3.4. Here X is the number slots each of size T_1 and Y is the cumulative of the number of rounds, each of size T_{ohd}, to complete the transmission of N packets using the selective repeat protocol. We shall ignore the constants T_1 and T_{ohd} in the following discussion and account for them only at the end. Since we have to compute $E[XY]$, we are interested in the joint distribution of the random variables X and Y. If $Y = R+1$, $R \geq 0$ and $X = N + k$, then k errors are distributed as $k_1, k_2, \cdots, k_R$, such that

$$N \geq k_1 \geq k_2 \geq \cdots \geq k_R > 0 \qquad (3.B.1)$$

The last (strict) inequality stresses the fact that all the $k_i's$ are greater than zero. The joint probability distribution of X and Y is given by

$$Pr[X = N + k, Y = R + 1] = \sum_{\{k_1+k_2+\cdots+kR=k\}} \binom{N}{k_1}\binom{k_1}{k_2}\cdots\binom{k_{R-1}}{kR}p^k q^N \qquad (3.B.2)$$

where the $k_i's$ satisfy the constraint in Equation 3.B.1.

Theorem 3.B.1:

$$\sum_{\{k_1+k_2+\cdots+kR=k\}} \binom{N}{k_1}\binom{k_1}{k_2}\cdots\binom{k_{R-1}}{kR}$$

is equal to the coefficient of x^k in $(1 + x + x^2 + \cdots + x^R)^N$ provided $N \geq k_1 \geq k_2 \geq \cdots \geq k_R \geq 0$ (notice that we are allowing the $k_i's$ to be zero here).

Proof:

By the binomial theorem,

$$(1 + x)^N = \sum_{k=0}^{N} \binom{N}{k} x^k$$

Substituting $x_1(1+x_2)$ for x in the above equation, we get

$$(1+x_1(1+x_2))^N = \sum_{k_1=0}^{N} \sum_{k_2=0}^{k_1} \binom{N}{k_1} \binom{k_1}{k_2} {x_1}^{k_1} {x_2}^{k_2} \qquad (3.B.3)$$

and putting $x_1 = x_2 = x$ we have

$$(1+x(1+x))^N = \sum_{k_1=0}^{N} \sum_{k_2=0}^{k_1} \binom{N}{k_1} \binom{k_1}{k_2} x^{k_1+k_2} \qquad (3.B.4)$$

The left hand side of Equation 3.B.4 equals $(1+x+x^2)^N$. Continuing this way, we can expand x_2 in Equation 3.B.3 to $x_2(1+x_3)$ and so on. This proves the theorem. ∎

In the above derivation, we have allowed the $k_i{}'s$ to be zero. This gives us $Pr[X = N+k, Y \leq R+1]$. Let $A(N+k,R) = Pr[X = N+k, Y \leq R+1]$ and $P(N+k,R) = Pr[X = N+k, Y = R+1]$. Then

$$P(N+k,R) = A(N+k,R) - A(N+k,R-1), \qquad R = 0,1,2,\cdots,k \quad (3.B.5)$$

Now,

$$(1+x+x^2+...+x^R)^N = \frac{(1-x^{R+1})^N}{(1-x)^N} \qquad (3.B.6)$$

If we denote the coefficient of x^k in Equation B.6 as $C(k,R)$, then

$$A(N+k,R) = C(k,R)p^k q^N \qquad (3.B.7)$$

Equations 3.B.5 and 3.B.7 finally give the probability of exactly $R+1$ rounds and k errors. Now we can compute

$$E[XY] = \sum_{R=0}^{\infty} \sum_{k=0}^{\infty} (R+1)(N+k)P(N+k,R) \qquad (3.B.8)$$

The covariance of X and Y is given by

$$cov(X,Y) = E[XY] - E[X]E[Y] \qquad (3.B.9)$$

E[X] and E[Y] have already been computed in Section 3.4. Equation 3.B.8 can be simplified as follows. Let $Q(XY, z, R)$ be defined as:

$$\begin{aligned} Q(XY,z,R) &= z^N(1+(pz)+(pz)^2+\cdots+(pz)^R)^N \\ &\quad - z^N(1+(pz)+(pz)^2+\cdots+(pz)^{R-1})^N \\ &= \frac{z^N}{(1-pz)^N}\left[\left(1-(pz)^{R+1}\right)^N - \left(1-(pz)^R\right)^N\right] \quad (3.B.10) \end{aligned}$$

On the other hand, from the definition of $P(N+k, R)$, and from equations 3.B.7, 3.B.8 and 3.B.10 we can see by inspection that

$$\begin{aligned} E[XY] &= q^N \sum_{R=0}^{\infty} (R+1) \frac{\partial}{\partial z} Q(XY,z,R) \Big|_{z=1} \\ &= q^N \frac{\partial}{\partial z} \sum_{R=0}^{\infty} (R+1) Q(XY,z,R) \Big|_{z=1} \quad (3.B.11) \end{aligned}$$

Using Equation 3.B.10, the inner sum on the right hand side becomes

$$\frac{z^N}{(1-pz)^N} \sum_{R=0}^{\infty} (R+1) \left(\left(1-\left(1-(pz)^{R+1}\right)^N\right) - \left(1-\left(1-(pz)^R\right)^N\right)\right)$$

Applying the formula for summation by parts to this expression, we get

$$\frac{z^N}{(1-pz)^N} \sum_{R=0}^{\infty} \left(1-\left(1-(pz)^R\right)^N\right)$$

Thus Equation 3.B.11 simplifies to

$$\begin{aligned} & q^N \frac{\partial}{\partial z} \left(\frac{z^N}{(1-pz)^N} \sum_{R=0}^{\infty} \left(1-\left(1-(pz)^R\right)^N\right)\right) \\ &= \frac{N}{q} \sum_{R=0}^{\infty} \left(1-\left(1-p^R\right)^N\right) + N \sum_{R=0}^{\infty} R\left(1-p^R\right)^{N-1} p^R \quad (3.B.12) \end{aligned}$$

The first term in Equation 3.B.12 can be seen from Section 3.4 to be equal to $E[X]E[Y]$. Hence, from equations 3.B.12 and 3.B.9, we have

$$cov(X,Y) = N\sum_{R=0}^{\infty} R(1-p^R)^{N-1}p^R \qquad (3.B.13)$$

For $Np << 1$, we can approximate this as

$$ov(X,Y) \approx N(1-p)^{N-1}p$$

and finally, putting $q = 1 - p$ we get

$$cov(X,Y) \approx Nq^{N-1}p \qquad (3.B.14)$$

Recall that the right hand side has to be multiplied by T_1T_{ohd} to finally give the correct covariance. In the range of interest, $cov(X,Y) \approx NpT_1T_{ohd}$ for small Np.

Appendix 5.A

We are interested in the variance of the time to absorption for a Continuous Time Markov Process, which starts off in a designated state i. Let

R_i = the time to absorption given we are in state i

t_i = sojourn time in state i

$H_i(s)$ = Laplace transform of t_i

$F_i(s)$ = Laplace transform of R_i

B = set of non-absorbing states

p_{ij} = Probability of going from state i to j in one step in the corresponding discrete chain.

Then, by the Markov property

$$F_i(s) = H_i(s)\left[\sum_{j\in B-i} p_{ij}F_j(s)\right] \tag{5.A.1}$$

Taking the natural logarithm of both sides, we have

$$\ln F_i(s) = \ln\left(H_i(s)\right) + \ln\left[\sum_{j\in B-i} p_{ij}F_j(s)\right] \tag{5.A.2}$$

Differentiating equation 5.A.2 and setting $s = 0$, we get,

$$E[R_i] = E[t_i] + \sum_{j\in B-i} p_{ij}E[R_j] \tag{5.A.3}$$

Differentiating equation 5.A.2 a second time, setting $s = 0$, we get, after some algebra:

$$Var(R_i) = Var(T_i) + \sum_{j\in B-i} p_{ij}Var(j) \tag{5.A.4}$$

$$+ \sum_{j\in B-i} p_{ij}(E[R_j])^2 - (E[R_i] - E[t_i])^2$$

which is the desired solution for the variance.

We also note that if i_0 is the initial state, then $E[R_{i_0}]$ gives the expected time to absorption for the Markov process of interest. This can be readily generalized if the initial distribution of the initial states are available.