Design and Analysis of Flow Control Algorithms for Data Networks

by

Paolo L. Narváez Guarnieri

S.B., Electrical Science and Engineering (1996)
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

Author .....
Department of Electrical Engineering and Computer Science
May 9, 1997

Certified by .....
Kai-Yeung (Sunny) Siu
Assistant Professor of Mechanical Engineering
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Design and Analysis of Flow Control Algorithms for Data Networks

by

Paolo L. Narváez Guarnieri

Submitted to the Department of Electrical Engineering and Computer Science
on May 9, 1997, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Flow control is a regulation policy which limits the source rates of a network so that the network is well utilized while minimizing traffic congestion and data loss. The difficulties in providing an effective flow control policy are caused by the burstiness of data traffic, the dynamic nature of the available bandwidth, as well as the feedback delay. This thesis develops analytical tools for designing efficient flow control algorithms for data networks.

Though the ideas presented in this thesis are general and apply to any virtual circuit networks, they will be explained in the context of the asynchronous transfer mode (ATM) technology. In particular, this thesis will focus on the available bit rate (ABR) service in ATM, which allows the use of network feedback in dynamically adjusting source rates.

Using a control theoretic framework, this thesis proposes a new design methodology for flow control algorithms based on separating the problem into two simpler components: rate reference control (RRC) and queue reference control (QRC). The RRC component matches the input rate of a switch to its available output bandwidth, while the QRC component maintains the queue level at a certain threshold and minimizes queue oscillation around that threshold. Separating the flow control problem into these two components decouples the complex dynamics of the system into two easier control problems which are analytically tractable. Moreover, the QRC component is shown to correct any errors computed by the RRC component by using measurements of the queue length, and thus can be viewed as an error observer. The interaction between the two components is analyzed, and the various ways in which they can be combined are discussed. These new theoretical insights allow us to design flow control algorithms that achieve fair rate allocations and high link utilization with small average queue sizes even in the presence of large delays.

This thesis also presents a new technique that simplifies the realization of flow control algorithms designed using the conceptual framework discussed above. The key idea is to simulate network dynamics using control cells in the same stream as the data cells, thereby implicitly implementing an observer. This "natural" observer eliminates

the need for explicitly measuring the round-trip delay. Incorporating these ideas, we describe a specific flow control algorithm that is provably stable and has the shortest possible transient response time. Moreover, the algorithm achieves fair bandwidth allocation among contending connections and maximizes network throughput. It also works for nodes with a FIFO queuing discipline by using the idea of virtual queuing.

The ideas discussed above assume that the flow control policy can be directly applied to the end systems. However, most data applications today are only connected to ATM via legacy networks, and whatever flow control mechanism is used in the ATM layer must terminate at the network interface. Recent results suggest that in these applications, ATM's flow control policy will only move the congestion point from the ATM network to the network interface. To address this issue, this thesis presents a new efficient scheme for regulating TCP traffic over ATM networks. The key idea is to match the TCP source rate to the ABR explicit rate by controlling the flow of TCP acknowledgments at network interfaces, thereby effectively extending the ABR flow control to the end systems. Analytical and simulation results are presented to show that the proposed scheme minimizes queue size without degrading the network throughput. Moreover, the scheme does not require any changes in TCP or ATM except at the network interface.

Thesis Supervisor: Kai-Yeung (Sunny) Siu
Title: Assistant Professor of Mechanical Engineering

Acknowledgments

First of all, I would like thank my research supervisor, Professor Kai-Yeung (Sunny) Siu, for his great support, guidance, and vision. The contents of this thesis are the result of many, many hours of discussions with him. I appreciate the freedom I have had to pursue my interests and his willingness to discuss new issues. His contributions to this work are immense.

Many thanks to the graduate students in my research group, Wenge Ren, Anthony Kam, Jason Hintersteiner, and Yuan Wu. Our daily interaction in the lab led to a constructive working environment. Thanks to all the professors and classmates I have met during my courses. The ideas and knowledge that I have learned from them will be part of me forever.

I also want to thank all the friends I have had in my five years at MIT. Without them, life at MIT would have been quite unbearable. Special thanks to Craig Barrack for all the discussions and helpful comments. He proofread much of the text in this thesis.

Finally, I want to express my deepest gratitude towards my parents for their unconditional support during all these years. Without them, none of this would have been possible for me. I also want to congratulate my sister, Marina, for her birthday two days from now.

Contents

List of Figures

Chapter 1

Introduction

1.1 Flow Control

Every data network has a limited amount of resources it can use to transport data from one host to another. These resources include link capacity, buffer, and processing speed at each node. Whenever the amount of data entering a node exceeds the service capacity of that node, the excess data must be stored in some buffer. However, the buffering capacity is also limited, and when the node runs out of free buffer space, any excess data is inevitably lost.

This over-utilization of resources is commonly referred to as *congestion.* As in many familiar examples (i.e. traffic in a freeway), congestion leads to a degradation of the overall performance. For example, if too many cars try to enter a freeway, the resulting traffic jam will cause every car to slow down. This in turn will cause the total rate of cars passing through a checkpoint to drop.

In data networks, congestion causes data to be lost at nodes with limited buffer. For most applications, the source will need to retransmit the lost data. Retransmission will add more traffic to the network and further aggravate the degree of congestion. Repeated retransmissions lower the effective throughput of the network.

In order to maximize the utilization of network resources, one must ensure that its nodes are subject to as little congestion as possible. In order to eliminate or reduce congestion, the rates at which the sources are sending data must be limited whenever

congestion does or is about to occur. However, over-limiting the source rates will also cause the network to be under-utilized.

Flow control is a traffic management policy which tries to ensure that the network is neither under-utilized nor congested. An efficient flow control scheme directly or indirectly causes the sources to send data at the correct rate so that the network is well utilized.

A good flow control scheme must also make sure that all the users are treated in a *fair* manner. When the rate of incoming data needs to be reduced, the flow control must limit the bandwidth of each source subject to a certain fairness criterion. Therefore, flow control also implies an allocation of bandwidth among the different sources that will guarantee not only efficiency but also fairness.

1.2 History

1.2.1 Transmission Control Protocol (TCP)

Even though the importance of flow control has been recognized since the first data networks, the problem did not become a critical issue until the mid 80's. At that time, it was realized that the Internet suffered from severe congestion crises which dramatically decreased the bandwidth of certain nodes [16]. In 1986, Van Jacobson's team made changes in the transmission control protocol (TCP) so that it effectively performed end-to-end flow control [9]. Ever since, newer versions of TCP (Tahoe, Reno, Vegas) have incorporated more sophisticated techniques to improve TCP flow control. Many of these improvements are based on intuitive ideas and their effectiveness has been demonstrated by extensive simulation results and years of operation in the real Internet.

The end-to-end flow control mechanism imposed by TCP is performed almost exclusively at the end systems. TCP makes no assumptions on how the network works and doesn't require the network to perform any flow control calculations. TCP flow control mechanism basically modifies the error recovery mechanism at the end

systems so that it can perform flow control as well. The detection of a lost packet is used by TCP as an indication of congestion in the network. Detecting lost packets causes the source to decrease its sending rate, whereas detecting no loss will cause the sending rate to increase. The disadvantage of this scheme is that congestion is dealt with only after it happens and after it is too late to prevent it. Furthermore, the loss of a packet does not necessarily indicate congestion, and the absence of loss does not indicate that the source can safely increase its rate. Under TCP's flow control scheme, a host or source has no idea of the state of the network other than the fact that packets are being lost or not. Valuable information about how much the links and queues are being utilized is simply not available to the transmitting source. Because of these reasons, TCP's end-to-end flow control, although functional and robust, is far from being optimal.

It is clear that in order to improve TCP flow control performance, we need to use a controller that uses more information about the current state of the network. Several researchers have proposed adding a flag in the TCP header that indicates congestion in the network [7, 18, 21]. The flag is turned on when the data passes through a congested node; the TCP source will then decrement the transmission rate. Other ways of improving TCP necessarily involve modifying TCP so that it can collect more information on the network status and then act accordingly.

At the same time, various underlying network technologies perform their own flow control schemes. For example, an ISDN link prevents congestion simply by allocating beforehand a fixed amount of bandwidth for every connection. If there is insufficient bandwidth, the admission control will simply reject attempts by new users to establish a connection. Some local area networks (LAN), such as token rings, have their own mechanisms to distribute bandwidth to the various users. Likewise, asynchronous transfer mode (ATM) networks have their own flow control schemes depending on the type of service offered. Because of ATM's flexibility and controllability, it is perhaps the easiest type of network on which to implement an efficient flow control scheme. In this thesis, we will mainly focus on how to design flow control algorithms in the more powerful ATM framework. However, the results will be shown to be

useful in performing flow control at the TCP layer as well.

1.2.2 Asynchronous Transfer Mode (ATM)

Asynchronous transfer mode (ATM) is generally considered the most promising network technology for future integrated communication services. The recent momentum behind the rapid standardization of ATM technology has come from data networking applications. As most data applications cannot predict their bandwidth requirements, they usually require a service that allows competing connections to share the available bandwidth. To satisfy such a service requirement, the ATM Forum (a de facto standardization organization for ATM) has defined the available bit rate (ABR) service to support data traffic over ATM networks.

When the ATM Forum initiated the work on ABR service, the idea was to design a simple flow control mechanism using feedback so that the sources can reduce or increase their rates on the basis of the current level of network congestion. As opposed to TCP flow control where the complexity of the controller resides in the end systems, ABR flow control performs most of its calculations in the network itself. To minimize the implementation cost, many of the early proposals used only a single bit in the data cells to indicate the feedback information. It has been realized that these single-bit algorithms cannot provide fast response to transient network congestion, and that more sophisticated explicit rate control algorithms are needed to yield satisfactory performance.

As the implementation cost in ATM switches continues to decline, many vendors are also looking for more efficient flow control algorithms. The difficulties in providing effective ABR service are due to the burstiness of data traffic, the dynamic nature of the available bandwidth, as well as the feedback delay. Though many proposed algorithms for ABR service have been shown to perform well empirically, their designs are mostly based on intuitive ideas and their performance characteristics have not been analyzed on a theoretical basis. For example, in the current standards for ABR service, quite a few parameters need to be set in the signaling phase to establish an ABR connection. It is commonly believed that ABR service performance can become

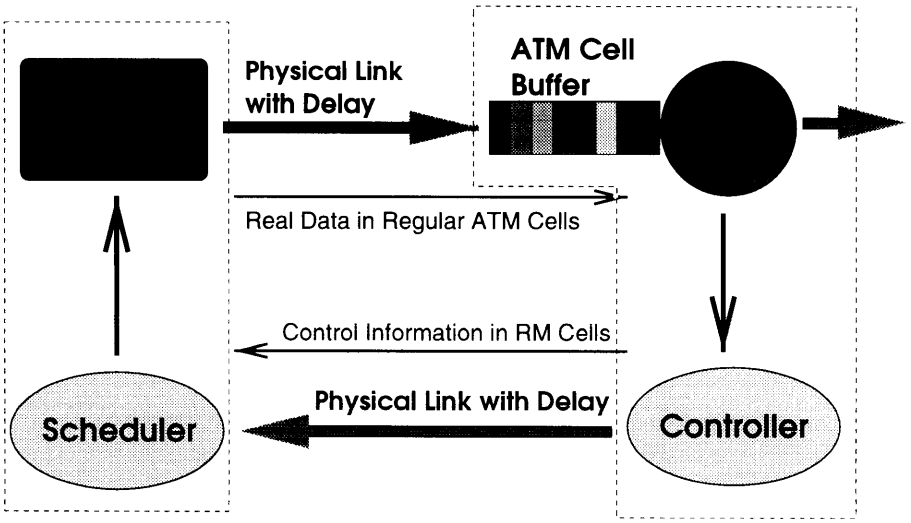

Figure 1-1: General ATM Flow Control Model

quite sensitive to changes in some of these parameters. However, how to tune these parameters to attain optimal performance is far from being understood, because of the complexity involved and the nonlinear relationship between the dynamic performance and changes in the parameters.

ABR service specifies a rate-based feedback mechanism for flow control. The general idea of the mechanism is to adjust the input rates of sources on the basis of the current level of congestion along their VCs. It requires network switches to constantly monitor the traffic load and feed the information back to the sources. Unlike the flow control approach used in TCP, ATM-ABR can explicitly calculate the desired incoming rates and directly command these rates from the sources. Figure 1-1 presents a single node model of the flow control mechanism in an ATM network. Data encapsulated in ATM cells travels through the physical links . At the switch, the cells are buffered before they get serviced. The novelty in the ABR service is that the node also contains a controller which calculates a desired transmission rate for the sources. This rate information encapsulated in resource management (RM) cells is then sent back to the source .

Recent interest in ABR service has brought about a vast amount of research activity on the design of feedback congestion control algorithms [1, 2, 8, 11, 12, 13,

14, 17, 23, 24]. They differ from one another by the various switch mechanisms employed to determine the congestion information and the mechanisms by which the source rates are adjusted. Depending on the specific switch mechanisms employed, various congestion control algorithms yield different transient behaviors.

In [2], classical control theory is used to model the closed loop dynamics of a congestion control algorithm. Using the analytical model, it addresses the problems of stability and fairness. However, as pointed out in [14], the algorithm is fairly complex and analytically intractable.

In [14], the congestion control problem is modeled by using only a first order system cascaded with a delay. The control algorithm is based on the idea of the Smith predictor [25], which is used as a tool to overcome the possible instabilities caused by the delays in the network. The Smith predictor heavily relies on knowledge of the round-trip delay. The objective of this algorithm in is to minimize cell loss due to buffer overflow. However, the algorithm only uses the queue level in determining the rate command, and without knowledge of the service rate, there is always a steady-state error with respect to the desired queue level. When the service rate changes (as is often the case in ABR service), the algorithm may lead to underutilization of the link capacity. The major limitation of the algorithm is that it assumes that the network delay is known and remains constant. If the actual delay differs from the assumed constant delay, the controller response will exhibit oscillations and in some cases instability.

This thesis uses many of the ideas presented in [14]. The Smith predictor is seen as a tool to estimate the amount of data in the links. This measurement is the key to being able to design *stable* flow control algorithms. Likewise, the queue stabilization algorithm presented in [14] is one of the two key controller components presented in this thesis.

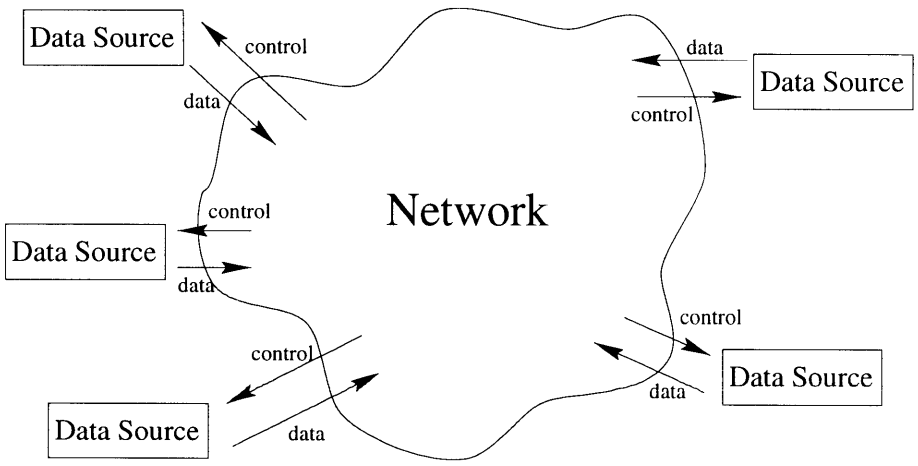

Figure 1-2: Network Flow Control Model

1.3 Objective

This thesis focuses on network-based flow control schemes. In these schemes, the network itself determines its level of congestion and determines how much data rate it is willing to accept from the sources. We assume that the network can communicate this desired rate to sources and that the sources will comply. Under this framework (as seen in figure 1-2), the network is directly controlling the source behavior. This type of network-based flow control is more powerful than those based on the end users (i.e. TCP flow control) because the information relevant to solving the flow control problem is present inside the network.

This approach is directly applicable in ATM's ABR service since the ATM specifications support this type of flow control. The switches can include their desired source rates in the explicit rate (ER) field contained in the RM cells returning to the source. The ATM source is then obliged to follow this explicit rate. At the same time, we would like to use the results in this thesis in an internetworking environment where the sources are not directly connected to ATM. For this purpose, we take advantage of the fact that most data applications use TCP in their transport layer, regardless of the network technology they are traversing. The latter part of this thesis explains how a network node can control the transmission rate of a TCP source. This technique effectively extends our network-based flow control algorithms to any source

that uses TCP as its transport protocol.

1.4 Overview

Using a control theoretic framework, this thesis studies various ways in which an effective flow control policy can be obtained. Because of its direct applicability, the flow control schemes will be presented in the context of ATM.

Chapter 2 will introduce some of the basic tools and ideas that can be used to enforce a flow control policy. Basic terminology such as available rate and effective queue is explained and simple flow control mechanisms are presented. The last section of the chapter proves some of the fundamental limitations involved in controlling a network.

Chapter 3 describes the basic flow control model used in this thesis. The model combines the basic techniques from chapter 2. The model is analyzed and refined until we arrive at the central idea of this thesis: the separation principle.

Under this principle, the flow control mechanism is separated into two key components: rate reference control (RRC) and queue reference control (QRC). The RRC component determines its rate command by using measurements of the available bandwidth in each link, with the objective of adapting the input source rate to match the output available rate at steady state. On the other hand, the QRC component computes its rate command by using measurements of the queue level at each switch, with the goal of maintaining the queue level at a certain threshold. The desirable source rate is then equal to the sum of the QRC and RRC rate commands. The advantage of this separation principle is that it decouples the complex dynamics of the closed-loop control system into two simpler control problems, which are analytically more tractable and can be solved independently.

Chapter 3 re-analyzes the separation principle from different viewpoints. The QRC component can be seen as an error observer of the RRC component. The chapter also explains how the separation principle can be implemented using a primal as well as a dual structure. The primal structure relies on the RRC to maintain the

steady-state command, while the dual structure relies on the QRC for this task.

In Chapter 4, various implementation issues are discussed. First, a discrete feedback mechanism is developed which ensures stable and fast closed-loop dynamics. The available rate is defined in the multi-user context and an algorithm is derived to estimate this available rate. The separation principle is extended to the multi-node case. Finally, both the single-node and multi-node cases are simulated.

In Chapter 5, a specific flow control algorithm called bi-channel rate control (BRC) is presented. Apart from using the separation principle and the other techniques from the previous chapters, the BRC algorithm uses virtual data and virtual queuing techniques. The virtual data technique creates a virtual channel in the network which is used for simulation purposes. This extra virtual channel allows the BRC algorithm to function without knowledge of the round-trip delay. The virtual queuing allows the algorithm to work in switches that use only FIFO queues. The new algorithm is simulated under the same conditions as the simulations in chapter 4.

Finally, in chapter 6 the TCP acknowledgment bucket scheme is presented. The idea behind this scheme is to manipulate the flow of TCP acknowledgments while they are traversing an ATM network so that the ATM network can control the rate of the TCP source. The scheme is implemented by using an acknowledgment bucket at the ATM network interface. Simulation results are also presented to demonstrate the effectiveness of the scheme.

Chapter 2

Basic Flow Control Ideas

In this chapter we will explore some of the basic ideas behind flow control algorithms. The purpose is to understand the fundamental difficulties one encounters as well as the various mechanisms that one can use in order to provide flow control. The first sections discuss various tools that can be used for the flow control problem. Using these tools, chapter 3 will expose a comprehensive model for our flow control system. In section 2.5, the fundamental time limitations of flow control are presented.

Our objective is to construct a controller that can regulate the flow of data entering the network. We assume that the network has the computational capability to calculate the correct distribution of rates and communicate these desired rates to the sources. We also assume that the sources will comply with the rate commands.

In order to compute the rate command, the switches can use any information that is available to them, such as queue sizes, service rates per VC, and any information sent by the users or other switches. The switches can also estimate other states in the network, such as the amount of data in the links.

2.1 Using Service Rate

The simplest mechanism to control the flow of user data into the network is to command the user to send data at a rate no greater than the service rate of the slowest switch in the virtual connection (VC). In steady state, the link is fully utilized and

there are no lost cells.

However, every time the service rate of the switch decreases, some cells can be lost. This inefficiency is due to the propagation delay in the links which prevents the source from reacting on time to changes in the service rates.

A major problem with this scheme is that the queue is unstable and cannot be controlled. In steady state, if there is any error in the service rate estimation, the queue will either always be empty or will grow until it overflows.

2.2 Using Queue Size

A slightly more complex flow control mechanism involves measuring the queue size at the switch. The switch computes a user data rate in order to fill the queue up to a reference queue size. This calculated rate is then sent as a command to the user. The computation of the rate requires a controller which outputs a rate command based on the error between the reference and real queue size. Since this control system attempts to stabilize the queue size to a certain level, it keeps the switch busy, maximizing throughput, while preventing cell loss.

However, as in the previous section, the propagation delays of the links prevent the source from reacting on time and can cause cell loss and unnecessary reduction of throughput. Furthermore, this system has the disadvantage that it can become unstable if the time delay or controller gain is sufficiently large.

2.3 Using Effective Queue Size

The instability of the preceding mechanism can be corrected by incorporating an effective queue into our controller. This effective queue includes all the data in the real queue of the switch, as well as the data that is traveling from the user to the switch and the data requests which are traveling from the switch to the source. The effective queue includes the number of cells that are bound to enter the real queue no matter what the switch commands in the future. Measurements or estimates of this

effective queue can be obtained using a Smith predictor or by using an accounting scheme. Both methods will be explained in the next section.

This control scheme is nearly identical to that in the previous section with the difference that the real queue size is replaced by the effective queue size. In this case, the controller tries to stabilize the effective queue size at a certain reference level. The advantage of this method is that since the switch can act on the effective queue instantaneously, there is no command delay and no instability due to such delay. In some sense, the system in which we are interested (queue) is modified so that it becomes easier to control.

The disadvantage of this technique is that the effective queue is not really what we are trying to control. When the data rate is smaller than the inverse of the link delay, the real queue is comparable to the effective queue and this approach is useful. However, as the data rate increases, the real queue becomes much smaller than the effective queue. At some point, in steady state, the real queue empties out and the data rate saturates. For high delay-bandwidth product connections, the data links are strongly under-utilized.

2.4 Estimating the Effective Queue Size

2.4.1 Accounting Estimator

One way of calculating the effective queue size is by using the accounting scheme. This is done by adding all the cells that the switch requests and subtracting all the cells that the switch receives. These cells that are "in flight" are then added to the real queue size, $q(t)$. If $u(t)$ is the rate command of the switch, $a(t)$ is the arrival rate of cells at the switch, the estimate of the effective queue size $q^\star(t)$ can be expressed as:

$$E_1[q^\star](t) = \int_{t_0}^{t} u(\tau)d\tau - \int_{t_0}^{t} a(\tau)d\tau + q(t) \tag{2.1}$$

This approach is similar to the credit based scheme. In fact, *credits* and $q^\star$ are related by: $credits = Maximum\ Queue\ Size - q^\star$. The disadvantage of this method is that it can be unstable. A disturbance in the number of cells that get sent would cause a permanent error in this estimate. For example, if a link is lossy and some percentage of the cells never make it to the queue, this estimator will add more cells than it subtracts, and the estimate will go to infinity.

2.4.2 Estimator with Smith Predictor

Another way of performing this calculation is by using a Smith predictor [25]. As in the previous method, the Smith predictor adds the number of cells that the switch requests and then subtracts them when they are expected to enter the switch's queue. This quantity is then added to the real queue's size. The calculation is done with the help of an artificial delay which figures out how many cells are about to enter the queue. If T_{est} is the estimated round-trip delay, the estimate of the effective queue is:

$$E_2[q^\star](t) = \int_{t_0}^{t} u(t)dt - \int_{t_0}^{t} u(\tau - T_{est})d\tau + q(t) \tag{2.2}$$

The advantage of this second approach is that it is robust against disturbances: a constant error in $u(t)$ will only produce a constant error in $q^\star(t)$. The disadvantage is that a good *a priori* estimate of the round-trip time is needed. If this estimate is incorrect or the real round-trip time varies, this estimator might produce a large error.

2.5 Transient Response and Performance

In this section, we will explore some of the fundamental limits on the transient response of flow control algorithms. We will show some of the restrictions that the network itself imposes on the controller.

We can think of a communication network as a system which contains states,

inputs, and disturbances. The states of this system are the sizes of the various queues, the number of cells in the channels, and the control information traveling in the links. The inputs are the rates at which the sources are sending data. The disturbances are the switch service rates and the number of virtual channels passing through each switch. The purpose of a flow control algorithm is to compute the correct inputs to this system so as to stabilize its states in spite of the disturbances.

If all the disturbances of a communication network (i.e. the number of VCs and switch output rates) are stationary, it is easy to establish a flow control policy. The correct rate for each connection can be calculated off-line so as to prevent congestion and ensure a max-min fair allocation of bandwidth. This fair distribution will ensure that the network operates indefinitely in steady state.

If some of the disturbances of the network system change in a completely *predictable* way, we can still calculate the correct allocations off-line and come up with the fair source rates to enforce at each moment of time. In this ideal case, the network states stabilize to a new value immediately after the change in the disturbance.

If the disturbances change in an *unpredictable* manner, it will take some time for the flow control algorithm to stabilize the network. This *settling time* is the time required for the flow control algorithm to *observe* changes in the system and to *control* them. In other words, it is the time to reach steady-state operation after the last change in the disturbances. The settling time is mainly determined by the network delays.

2.5.1 Single Node Network

When the network contains only one node, we can obtain very strong bounds on the settling time. The settling time varies depending on whether the delay of the link between the source and the switch is known or not. The minimum bounds on the time needed to stabilize such systems are stated in the following theorems:

Theorem 1 For a network consisting of a single source and a single switch connected by a link with a *known* delay, the minimum time in which *any* flow control algorithm

can stabilize the network after a change in the switch's service rate is **the round-trip delay time (RTD)** between the source and the switch.

Proof: Let us assume that at time zero the last change in the switch's service rate takes place. Because the link delay is known, the controller can calculate the effects of this change on the network as soon as it knows of the change. Once the future effects are known, the controller can request the source to send data at a rate that will correct the effects of the change of the service rate and immediately after at a rate that will reestablish steady-state operation. If the controller is located at the source, it will know of the service rate change at 1/2 RTD. If the controller is in the switch, it can send its new rate requests at time 0, but the requests will only reach the source at time 1/2 RTD. In both cases, the source starts sending data at the new stable rate at time 1/2 RTD, which will only reach the switch at time 1 RTD. □

Note that this lower bound can only be achieved by a flow control algorithm that can accurately measure the disturbance (switch service rate), predict its effect on the network, and request new rates instantly that will reinstate the switch's queue size.

To prove the next theorem, we first need to prove the following lemma:

Lemma 1 For a network consisting of a single source and a single switch connected by a link with an *unknown* delay, the minimum time needed to detect the effect of a change in the switch's service rate (given that the queue does not saturate) is the round-trip delay of the link (1 RTD).

Proof: At the moment when the last change of the switch's service rate takes place (time 0), the switch's output queue is changing (increasing or decreasing). Regardless of what the controller does after time 0, the queue will continue to change in an uncontrollable manner for 1 RTD. This is the time that it takes for a new controller command to have an effect on the queue. Since the RTD is unknown, between time 0 and RTD, the controller does not know how much the queue size will change before its command has an effect on the queue. During this "unpredictable" period, the controller cannot know what the effect of the service rate change will be on the queue

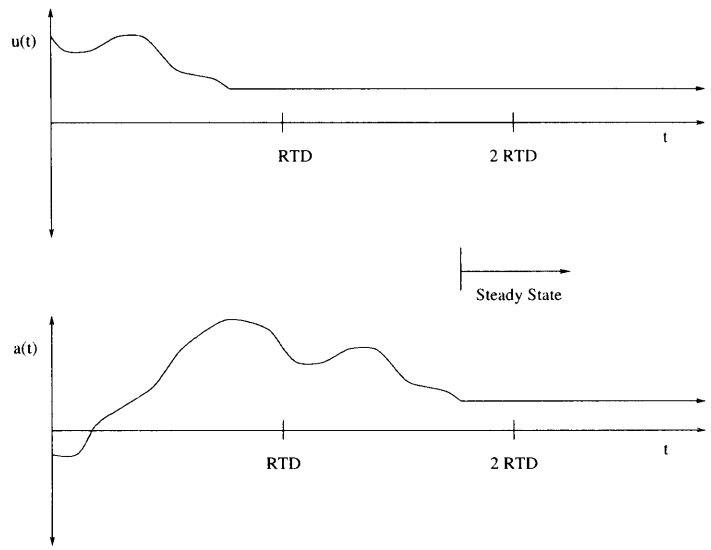

Figure 2-1: Network with Unachievable Settling Time

size. It is only after time RTD that the controller can *possibly* know what the full consequence of the disturbance is on the queue size. The only exception is when the queue size saturates before time RTD; in this case, the controller knows the effect of the rate change when the saturation occurs. □

Theorem 2 For a network consisting of a single source and a single switch connected by a link with an *unknown* delay, the minimum time in which *any* flow control algorithm can stabilize the network after a change in the switch's service rate , given that the queue does not saturate, is **twice the round-trip delay time (2 RTD)** between the source and the switch.

Proof: Let us assume on the contrary that after the last change of the switch's service rate at time 0, the network can be stabilized before time 2 RTD. Figure 2-1 shows an example of this scenario. In this case, the controller is located in the switch. Let $u(t)$ be the instantaneous rate command given by the controller and $a(t)$ be the arrival rate of data into the switch. Since we assumed that the system reaches steady state before 2 RTD, $a(t)$ must reach steady state before 2 RTD (otherwise the queue size would oscillate). Because the source complies with the controller's command, we

know that $a(t) = u(t-\text{RTD})$. Therefore, $u(t)$ must reach steady state before 1 RTD. This implies that the controller must know what the effect of the rate change at time 0 is before 1 RTD. From Lemma 1 we know that this is not possible if the queue does not saturate. Therefore, our original assumption must be wrong, which proves the theorem. The same argument can be applied if the controller is located in the source or any other place. □

Note that we have not proved that it is possible to stabilize the network in 2 RTD, but that it is impossible to do so in less than that time. From a control-theoretic viewpoint, we can interpret this lower bound as the algorithm needing at least 1 RTD to *observe* the network and 1 RTD to *control* it.

2.5.2 Multiple Node Network

The transient response time in a multi-node network is much harder to analyze than in the single-node case. If some change of disturbance takes place in a particular node, it will trigger changes in the states of other nodes. These changes might propagate to the rest of the network.

The analysis is greatly simplified if we assume that there is no communication or collaboration between the different VCs in the network. From the perspective of one particular VC (which we will call VC i, the activities of the other VCs can be regarded as disturbances on the queue sizes and available bandwidth of VC i on each of its nodes. Furthermore, if the switches use a per-VC queuing discipline, the queue size of each VC will not be directly affected by the other VCs; the only disturbance apparent to VC i is the available bandwidth at each of its nodes.

We will assume that the VCs do not collaborate with each other. For the moment, we will also assume that the switches use a per-VC queuing discipline. Under these assumptions, we can determine the minimum time to achieve steady state in one particular node in a VC after a change in its available rate, given that the available rate at the other nodes in the VC are already in steady state. It is the time it takes for a control data unit that departs the node during the node's last change of

disturbance to travel to the source and back twice in a row. This is what we will refer to as "2 RTD" of a particular node in a multi-node network. The time to complete the first loop is the minimum time needed to observe the effects of the last change of the disturbance on the VC, while the time to complete the second loop is the minimum time to control the VC queue at the node. This limit can be proved with the same arguments as in the proof of the single node case. The multi-node 2 RTD is determined by all the queuing delays upstream as well as by the propagation delays of the links.

Note that we have not shown anything about the global transient responses and stability of the entire network. We are just showing what is the shortest settling time possible for one VC if the other VCs are already in steady state.

Chapter 3

General Single Node Model and Analysis

We will start our analysis by creating a general model which can use all the flow control schemes discussed in the previous chapter. The model is limited to the single node case. This model is a generalization of the one presented in [14]. The model uses a fluid approximation for all the data traffic and control signals.

On the left side of the model represented in figure 3-1, we can observe the control system that sends the rate command $u(t)$ to the source. This command is sent to the source and returns as an actual data rate. The feedback and data channel are modeled as a delay e^{-sT} where T is the round-trip delay time. The rate of incoming data is integrated to obtain the real queue size $q(t)$. At the same time, the integral of the service rate $r(t)$ is subtracted out of the queue size to reflect that the queue is being serviced.

The rate command $u(t)$ is the sum of the commands $u_r(t)$ and $u_q(t)$, which are based on the service rate and effective queue size information respectively. We will refer to the subsystems with outputs $u_q(t)$ and $u_r(t)$ as *queue reference control (QRC)* and *rate reference control (RRC)* respectively.

The QRC contains an observer which estimates the effective queue size. This estimate is compared to the desired queue size $x(t)$ and the error is fed into a controller $K(s)$ which determines the correct rate u_q to decrease the error. The RRC contains

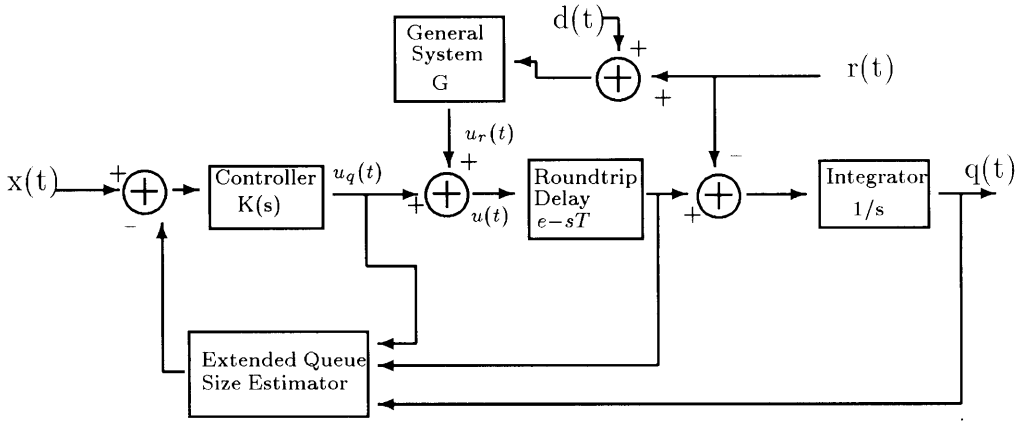

Figure 3-1: General Model of Flow Control System

a system G (not necessarily linear) which takes as an input an approximation of the service rate $r_{app}(t)$ and outputs the RRC part of the source rate $u_r(t)$. The general model of $r_{app}(t)$ is some disturbance $d(t)$ added to the real service rate $r(t)$.

This model allows us to use any combination of methods described in the last chapter. The main innovation in this model is that it allows for the combination of subsystems that can use both the service rate and the effective queue size as control information.

3.1 Primal Control Model

We have seen in the previous chapter that under ideal circumstances, the smith predictor and the accounting system should have the same behavior in estimating the effective queue size. In isolated environments, QRC controllers that use a Smith predictor or a an accounting system have identical dynamics (given that all parameters are correct). However when the RRC is added to the system, the overall system dynamics are different for each case. For the moment, we will concentrate in describing the behavior of the system using a Smith predictor as shown in figure 3-2. We call call this model the *primal* control model. Later on, in section 3.3, we will show how to modify and reanalyze the system for when it uses an accounting system rather than a Smith predictor.

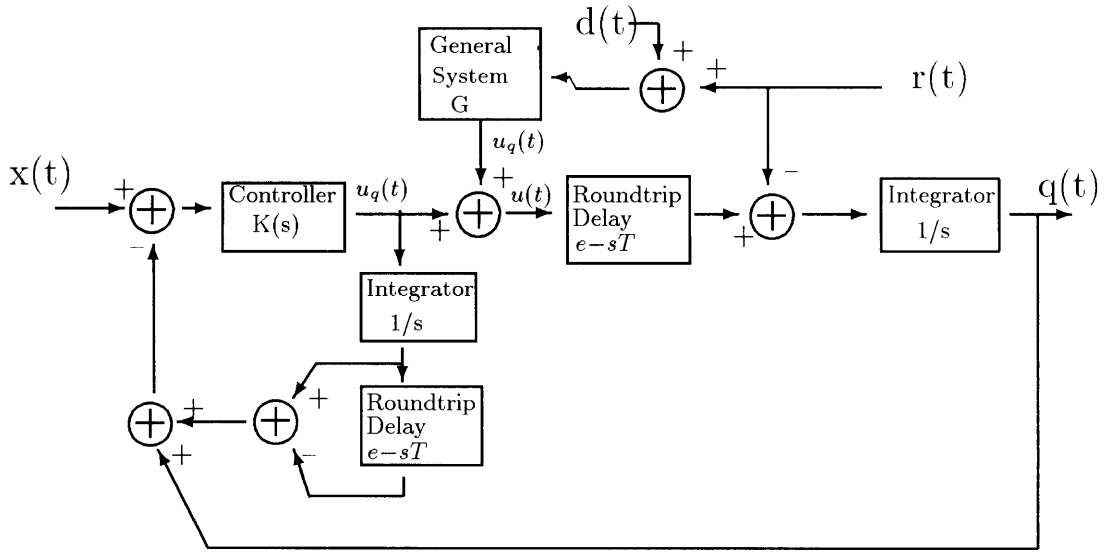

Figure 3-2: Primal Control Model

3.1.1 Input/Output Relations

For the moment, let us also assume that the subsystem G is linear time-invariant (LTI). Now, the complete continuous time model is LTI. Its dynamics can be completely characterized in terms of the transfer functions between its inputs and outputs. The relevant inputs are: *x(t)*, *r(t)*, and *d(t)*. The only relevant output is *q(t)*,the queue size.

The transfer functions between the inputs *x(t)*, *r(t)*, *d(t)* and the output *q(t)* are:

$$H_x(s) = \frac{Q(s)}{X(s)} = \frac{K(s)e^{-sT}}{s + K(s)} \tag{3.1}$$

$$H_r(s) = \frac{Q(s)}{R(s)} = \frac{G(s)e^{-sT} - 1}{s + K(s)} \left[1 + \frac{K(s)}{s}\left(1 - e^{-sT}\right)\right] \tag{3.2}$$

$$H_d(s) = \frac{1 + \frac{K(s)}{s}\left(1 - e^{-sT}\right)}{s + K(s)} G(s)e^{-sT} \ . \tag{3.3}$$

With these transfer functions, we can completely determine q(t) from the inputs. In the following subsection we will describe the system's response to each of the inputs. Note that the total response is the superposition of the response to each input: $q(t) = q_x(t) + q_r(t) + q_d(t)$.

3.1.2 Response to Desired Queue Size

First of all, we would like $q_x(t)$ to follow $x(t)$ as closely and as fast as possible. In steady state (if $x(t)$ is a constant X_0), we would like $q_x(t) = x(t) = X_0$. In terms of the the transfer function shown in equation (3.1), this means that $H_x(s)$ should be one for $s = 0$. A good response to a change in $x(t)$ requires that $H_x(s)$ be close to one at least for small s.

A simple and efficient way of obtaining these results is to set $K(s)$ to a constant k. The simplified transfer function is:

$$H_x(s) = \frac{k}{s+k} e^{-sT} \ . \tag{3.4}$$

When k is large, $q_x(t)$ follows $x(t)$ closely for low frequencies (small s). Furthermore, the response is always a first order exponential followed by a delay. This means that there are no unwanted oscillations and the output is always stable (for this input). A larger k will offer a better response. However, we will later see that k is limited by the frequency of the feedback.

3.1.3 Response to the Service Rate

Now, we will examine how $q_r(t)$ varies with $r(t)$. This knowledge is of particular importance because it tells us what type of service rate behavior will make the queue overflow or empty out. This calculation can be accomplished by convolving the impulse response of the system with $r(t)$. For the moment, we assume that G is a constant g for all s. The queue's response $q_r(t)$ to an impulse in the switch rate $r(t)$ simplifies to:

$$h_r(t) = -u(t) + \left(1 + g - e^{-k(t-T)}\right) u(t-T) - \left(ge^{-k(t-2T)} - g\right) u(t-2T) \ , \tag{3.5}$$

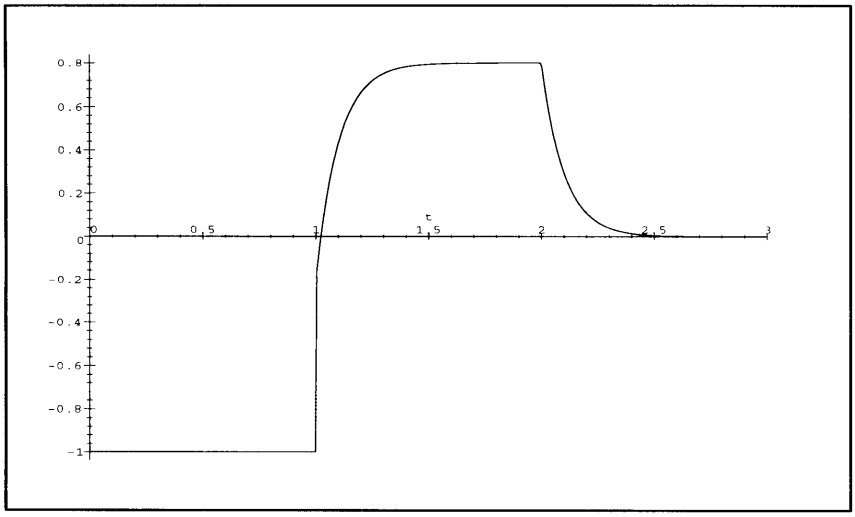

Figure 3-3: Impulse Response $h_r(t)$ with k=10, g=0.8, T=1

where $u(t)$ is the unit step function.

This impulse response with a given set of parameters is illustrated in figure 3-3. For large enough gain k, this impulse response has the nice property of approximating two contiguous boxes with width T and heights -1 and g. Therefore, the queue size response caused by $r(t)$ at any time t can be approximated by:

$$q_r(t) \approx g \int_{t-2T}^{t-T} r(\tau)d\tau - \int_{t-T}^{t} r(\tau)d\tau \ . \tag{3.6}$$

When the approximation is true, we only need information on $r(t)$ from $t-2T$ to t to obtain $q_r(t)$. If the service rate does not change much relative to the round-trip time T, this calculation is easy to compute.

This approximation shows how instantaneous $q(t)$ does not depend on on the instantaneous $r(t)$, but rather on $r(t)$ averaged over time intervals $t-2T$ to $t-T$ and $t-T$ to t. This means that if $r(t)$ oscillates wildly at very high frequencies, the queue size will not be affected as long as the time averaged $q(t)$ is reasonably stable.

When $g = 0$, $q_r(t)$ is always negative, meaning that $q(t) < q_x \approx x(t)$. If $x(t)$ is less than the maximum queue size, we have the certainty that the queue will not overflow.

This is the scheme proposed in [14]. Its disadvantage is that in steady state there is a difference between the desired queue size $x(t)$ and the actual queue size $q(t)$. For high rates, the queue will empty out and the link will not be fully utilized.

If $g = 1$, $q_r(t)$ can be either positive or negative. The actual queue can either overflow or empty out. The advantage of this scheme is that in steady state ($r(t)$ is constant), the queue level $q(t)$ is always equal to the desired level $x(t)$. In this case, the queue is stable and the link is fully utilized.

If $0 < g < 1$, we can obtain a a queue behavior that is a weighted average of the two previous ones. If the delay-bandwidth product is low ($r(t) \times T << Max\ Queue\ Size$) on average, we might set g to be closer to zero, since we are not losing much throughput and we can obtain a low cell loss performance. On the other hand, if the delay-bandwidth product is high on average, we might set g to be closer to one, since that will increase the throughput, even if the risk of losing cells is higher. Likewise, if we want cell loss to be minimized (i.e. for data transfer applications), we choose g small. If throughput is more important than occasional cell loss (i.e. for video transmissions), we choose a large g.

Step Response

In a real system, most of the changes in the service rate will be step changes. The service rate will be constant during certain intervals of time and will change abruptly when there is a change in the operation of the network, i.e. a new user connects, other services like VBR require bandwidth from ABR. By using Laplace transform methods, we can analytically determine how $q_r(t)$ will react to a unit step change in $r(t)$:

$$q_r(t) = -tu(t) + \left[(1+g)(t-T) - \frac{1-e^{-k(t-T)}}{k}\right] u(t-T) - \left[g(t-2T) - \frac{g(1-e^{-k(t-2T)})}{k}\right] u(t-2T)\ . \tag{3.7}$$

In figure 3-4, we can visualize how $q_r(t)$ reacts to a unit step change in $r(t)$ for

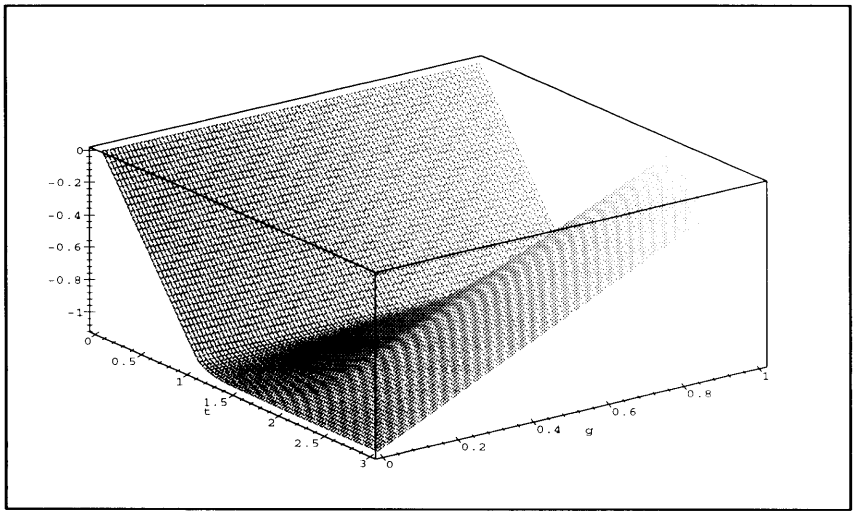

Figure 3-4: Step Response $h_r(t)$ for k=10,T=1

different values of g. Between time 0 and T, the system's new command has not traversed the delay yet, and $q_r(t)$ keeps moving away from its previous value. At time T, the system starts reacting with different strengths depending on the value of g. For $g = 0$, $q_r(t)$ stabilizes at T and does not recover from the change between 0 and T. For $g = 1$, $q_r(t)$ is restored at time $2T$ to its previous value, and the system stabilizes from there onwards. For $0 < g < 1$, $q_r(t)$ only partially recovers its previous value. The response of a greater or smaller step is simply scaled appropriately.

Frequency Response

Another interesting characteristic of the system is the frequency response between $r(t)$ and $q_r(t)$. The frequency response describes how $q_r(t)$ reacts to sinusoidal inputs of $r(t)$ with different frequencies w and unit amplitude. In figure 3-5, we can see this frequency response with variable parameter g. A greater response for a particular w means that $q_r(t)$ follows $r(t)$ with a higher amplitude.

Ideally, we would like the frequency response to be zero for all w. This would mean that the queue does not react to the service rate. When $g = 1$ this is true for $w = 0$, which is equivalent to saying that there is no steady-state response. For smaller g, there appears some response at $w = 0$, and this response increases as g

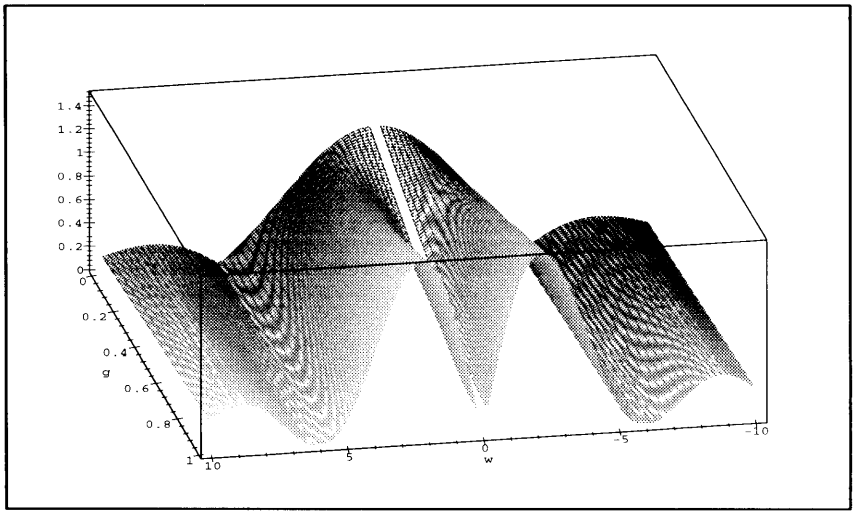

Figure 3-5: Frequency Response $|H_r(s)|$ for k=10,T=1

gets smaller. However, when g is made smaller, the maximum frequency response (highest peak) decreases.

As mentioned earlier in this section, the response of the queue to oscillations of $r(t)$ becomes negligible as the frequency of $r(t)$ increases. The most perturbing frequencies are the ones comparable to the inverse of the delay time ($w \approx \pi/T$).

3.1.4 Error Sensitivity

We will finally analyze the system's sensitivity to an error $d(t)$ in the measurement of the service rate. If an algorithm is purely RRC ($k = 0$), any finite $d(t)$ can cause $q_d(t)$ and the entire queue to be unstable (overflow or empty out). However, when QRC is added to RRC, as in our model, $q_d(t)$ becomes stable.

From equations (3.2) and (3.3), we can see how the response to $d(t)$ for any constant $G(s)$ is proportional to the delayed part of the response to $r(t)$. Both input/output relations are stable. This means that a finite $d(t)$ will only cause a finite $q_d(t)$. Therefore, a finite error in the rate measurement will only cause a finite error in the real queue size.

3.2 Model Discussion and Improvements

3.2.1 Examining QRC and RRC Independently

Let us examine the dynamics between $r(t)$ and $q(t)$ when either $K(s)$ is zero and $G(s)$ is one or $G(s)$ is zero and $K(s)$ is large. These two situations correspond to using only the service rate (RRC) or the queue size (QRC) respectively as control references. From equation (3.5), we can see that the impulse response from $r(t)$ to $q(t)$ in each situation becomes:

$$h_r^{RRC} = -u(t) + u(t-T) \tag{3.8}$$

$$\begin{aligned} h_r^{QRC} &= -u(t) + (1 - e^{-k(t-T)})u(t-T) \\ &\approx -u(t) + u(t-T)\ . \end{aligned} \tag{3.9}$$

For large k, each type of control has the same response to changes in the service rate. In this case, the dynamics of QRC and RRC working independently are identical. Both algorithms are effective in utilizing the link bandwidth while the queue size is between its limits. However, both algorithms are ineffectual at stabilizing the queue size. In steady state as $r(t)$ increases, $q(t)$ decreases. As can be seen in figure 3-6, for a step change in $r(t)$, the most each type of algorithm can do is to stop $q(t)$ from changing after time T. Neither of them alone can recover from the change of $q(t)$ between time 0 and T. From the same figure, we can see that the delay T in the response of RRC or QRC (with large k) to $r(t)$ causes a change in the queue size $\Delta q(t)$ equal to the shaded area between the $r(t)$ and $u(t-T)$ curves.

The difference between the two independent controls arises when the system reaches the limits of the linearized model. The linearity of the model does not hold any longer once the queue is emptied out and more cells cannot be serviced , or when the queue overflows and more cells cannot enter the queue. In these cases, QRC behaves by freezing the rate command $u(t)$, whereas RRC continues moving $u(t)$ towards $r(t)$. When the queue size is again within its limits and the linear regime is

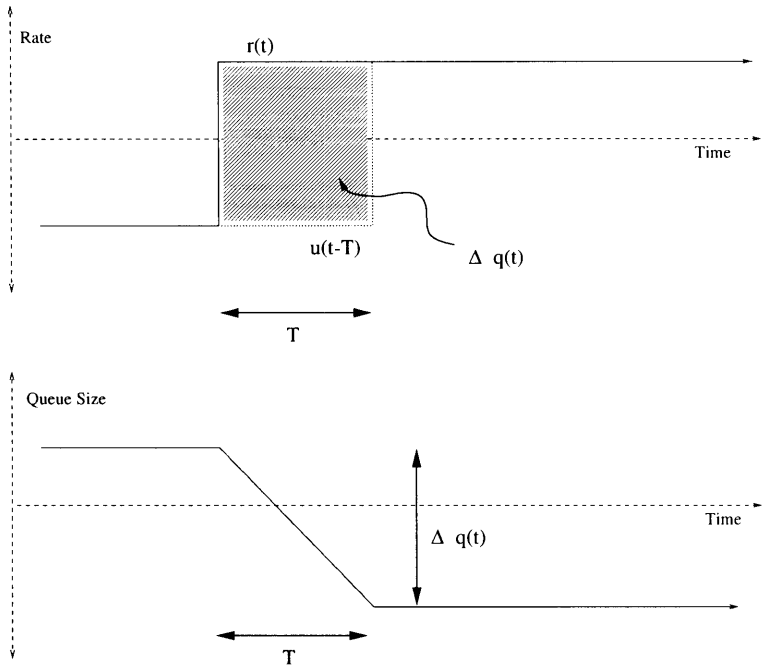

Figure 3-6: Step Response of QRC or RRC as Isolated Systems

reestablished, QRC will be operating in the same region as before, while RRC will be operating in another region. In this case, the queue size resulting from RRC will have a constant offset from the corresponding command resulting from QRC. In general, a QRC algorithm will always generate the same queue size for the same constant service rate, while the RRC algorithm will generate a queue size that also depends on how many cells have been lost (full queue) or have not been utilized (empty queue).

Independent QRC and RRC each have a window of linear operation which limits the rate command it can issue when the system is in the linear regime. If the system is initialized with no service rate ($r(t) = 0$) and a constant queue reference ($x(t) = X_0$), the initial window of operation is constrained by:

$$u_{min} = \frac{X_0 - Q_{max}}{T} < u(t) < u_{max} = \frac{X_0}{T} , \tag{3.10}$$

where Q_{max} is the maximum queue size. When $r(t)$ is beyond these limits, the system enters the non-linear regime. If $r(t) < u_{min}$, cells are lost due to queue overflow, and if $r(t) > u_{max}$ service slots are not being used.

In the case of QRC, its window is fixed, and $u(t)$ always has the same constraints. On the other hand, for RRC, the linear operation window slides after delay T so that $r(t)$ will always be inside the window. The width of the window is constant (Q_{max}/T), but the edges will move to follow $r(t)$ if it is operating in the non-linear regime.

For QRC, if $X_0 < Q_{max}$, $r(t)$ can never be less than u_{min} (since u_{min} is negative). Therefore, the queue cannot overflow. In fact, the queue size of an independent QRC will always be less than X_0. For this reason, independent QRC is a good control scheme for scenarios where we absolutely do not want cell loss. On the other hand, when $r(t) > u_{max}$, QRC will be continuously missing service opportunities.

In RRC, the window of operation takes T time to follow $r(t)$. During this delay, independent RRC will lose cells or miss service slots. However, after T, RRC will be back in the linear regime and fully utilize the link with no loss. For this reason, independent RRC is a good control choice when $r(t)$ does not vary much and we want full utilization of the available links.

3.2.2 Superposition of QRC and RRC

Many of the problems and flaws of QRC and RRC can be corrected by making the two work together. In our original model this is equivalent to adding the output of $G(s) = 1$ to the output of $K(s)$ to obtain $u(t)$. When this happens, the desired real queue reference level is always attainable and throughput is always maximized. The two algorithms are complementary in nature, and one can correct the defects of the other.

In figure 3-7, we can observe the behavior of the cell arrival rate $a(t) = u(t - T)$ of the combined controller (with k large) after a step in $r(t)$. For a time interval of T, the system does not react, and the queue size changes by an amount equal to the first shaded area. However, between time T and $2T$, $a(t)$ over-reacts by the right amount so that the previous queue level is restored by time $2T$. Notice how the area of the second shaded region is the negative of that of the first region. After time $2T$, $a(t)$ is equal to $q(t)$ and the queue size and the system in general are stable.

From QRC's perspective, we can think of RRC as modifying the apparent service

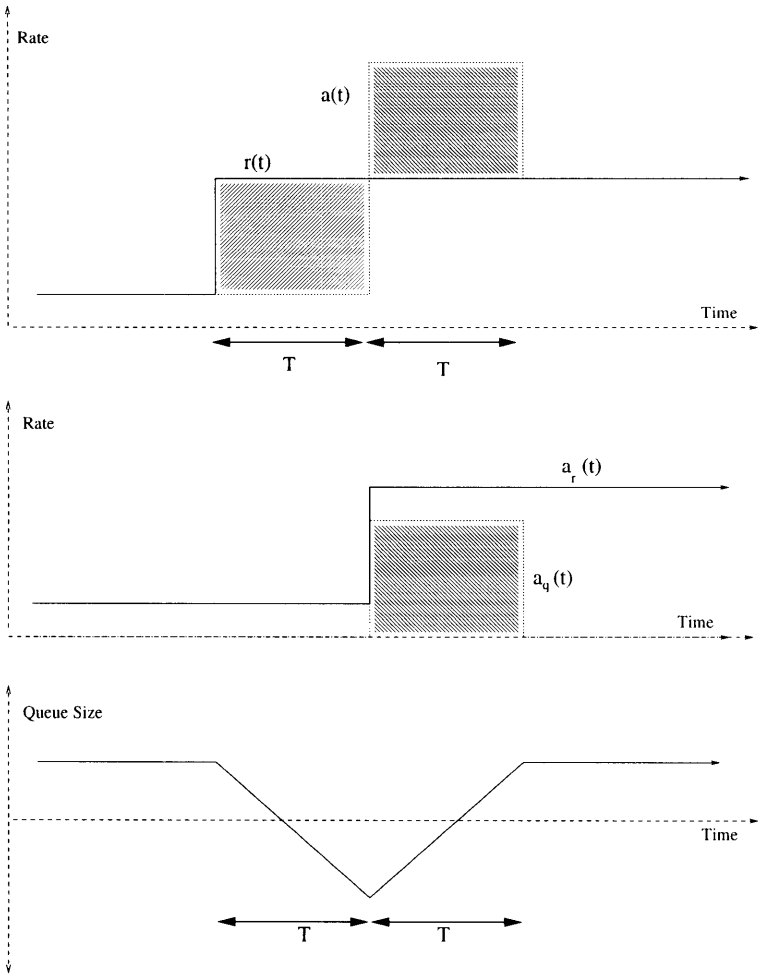

Figure 3-7: Step Response of the Superposition of QRC and RRC

rate to $r_{app} = r(t) - u_r(t-T)$. This means that the average apparent rate that QRC sees is zero. QRC only sees an apparent rate during transients. QRC then cancels the effects on the queue of these transient apparent rates. QRC sees RRC as a tool to eliminate the DC-level service rate which causes the effective queue size to be bigger than the real queue size. In this way, the reference level, $x(t)$ commands the *real* queue size and not just the *effective* queue size.

From RRC's perspective, we can think of QRC as the feedback needed to correct its previous outputs. This can be seen in figure 3-7 where between time T and $2T$, QRC causes an arrival rate $a_q(t)$ equal to the difference between $r(t)$ and the arrival rate due to RRC $a_r(t)$ between time 0 and T. In general, the arrival rate due to QRC is approximately

$$a_q(t) \approx r(t-T) - a_r(t-T) \ . \tag{3.11}$$

This equation shows how QRC corrects the error of RRC after time T

The superposition of RRC and QRC also creates a new linear operating window which borrows good characteristics from both RRC's sliding window and QRC's fixed window. The new window slides time T after $r(t)$ changes. The position of the window is fixed relative to $r(t)$. The edges of the new window are always:

$$a_{min}(t) = u_{min}(t-T) = r(t-T) + \frac{X_0 - Q_{max}}{T} \tag{3.12}$$

$$a_{max}(t) = u_{max}(t-T) = r(t-T) + \frac{X_0}{T} \ . \tag{3.13}$$

Under the combined RRC + QRC scheme it is possible to slide the window of operation without leaving the linear regime. This means that if $r(t)$ varies slowly enough, $a(t)$ will follow $r(t)$ closely enough so that there will be no cell loss or missed service slots. Using the window argument, the step change of the service rate that will guarantee no cell loss or no missed service slots are:

$$\text{No Cell Loss:} \quad \Delta r(t) > -\frac{Q_{max} - X_0}{T} \tag{3.14}$$

$$\text{No Unserviced Cells:} \quad \Delta r(t) < \frac{X_0}{T} \, . \tag{3.15}$$

By choosing X_0, we can adjust these "safety margins". For example, by choosing $X_0 = 0$, we can maximize the negative range of $\Delta r(t)$ and therefore minimize the probability of cell loss.

3.2.3 QRC as an Error Observer

Another interpretation that can be given to QRC is that of an observer which estimates the error of RRC using only measurements of the real queue size.

As we mentioned previously, the RRC algorithm working alone tries to match the arrival rate $a(t)$ with the service rate $r(t)$, but is alway one round-trip delay time T late. This delay causes the queue size to increase or decrease for T amount of time. We will define the rate of change in the queue size due to the RRC as $e(t)$, the error of the RRC algorithm working alone. Ideally, we want $e(t)$ to be zero at least on average. This can be accomplished by estimating $e(t)$ and subtracting it from the command of RRC $u_r(t)$. In other words, we can correct $e(t)$ by creating a command $-e(t)$ at a later time.

In order to estimate $e(t)$ we cannot simply differentiate the queue size $q(t)$ since that will give us the rate of change of $q(t)$ due to the RRC and its correction. We need to know the rate of change due to the RRC only. We can obtain this quantity artificially through an observer that simulates the RRC and the the system. The observer can also use feedback from the queue size to ensure that its states and variables will converge with those of the real system.

In figure 3-8, we can follow the different stages of how an observer for $e(t)$ can be built, and how the final design is equivalent to a QRC system. The first stage shows how to make an $e(t)$ observer that is independent of the RRC. The second stage shows how to modify the observer so that $-e(t)$ can be be added to the rate

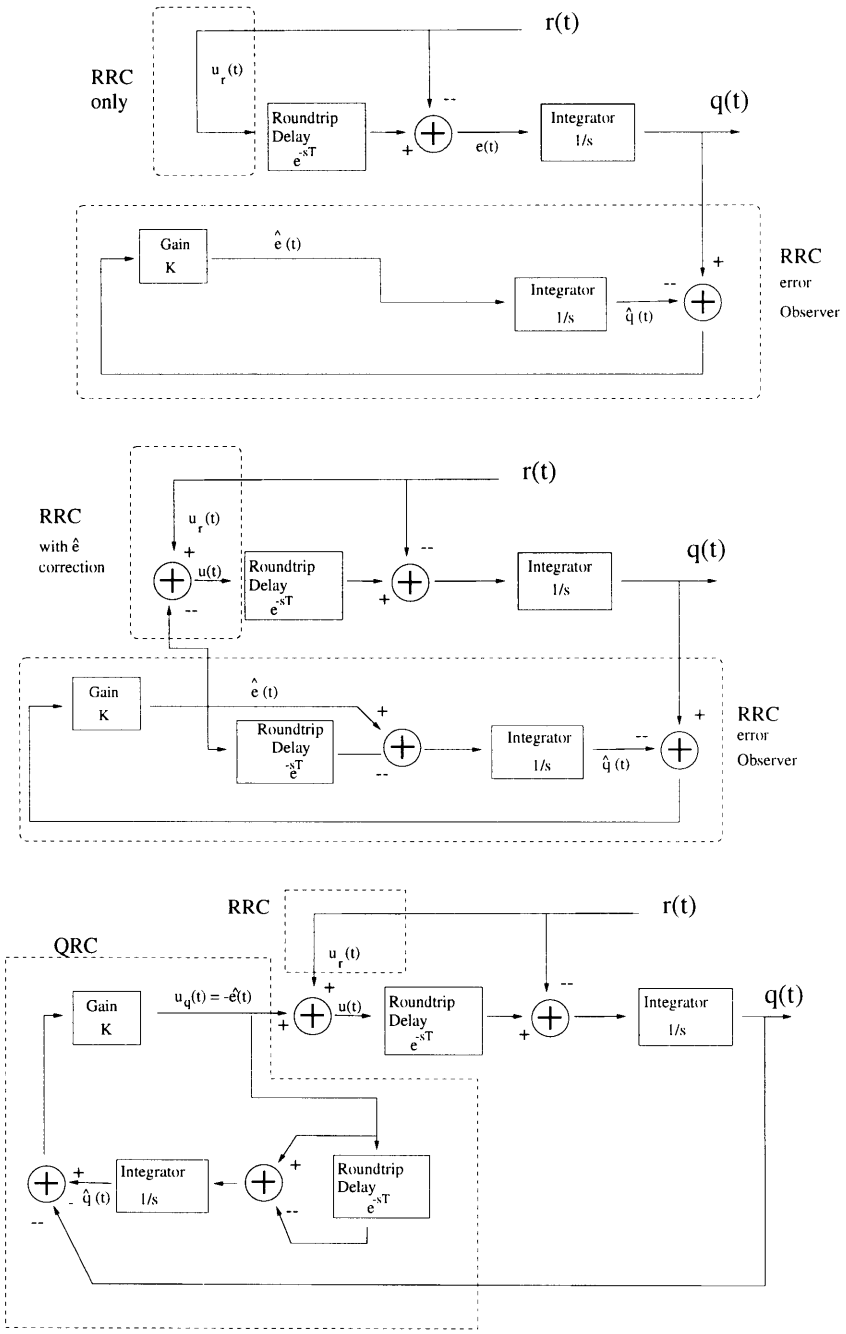

Figure 3-8: QRC is an Observer of RRC's Error

command in order to correct the RRC. The third stage is another way of redrawing the second stage. It shows how the RRC with error observer and correction displayed in the second stage is equivalent to the superposition of RRC and QRC.

3.2.4 Linear Control with Weighted RRC and QRC

In the previous sections, we discussed the performance of the full superposition of the RRC and QRC components. The combination is able to maximize throughput for any $r(t)$. However, when $r(t)$ changes very rapidly, even with $X_0 = 0$, the safety margin might not be large enough to prevent cell loss. The safety margin can be increased by decreasing the role of RRC in the overall control. In our model this can be done by making g smaller than one.

The maximum step decrease to prevent cell loss becomes:

$$-\Delta r(t) < (1-g)r(t) + \frac{Q_{max} - X_0}{T} \ . \tag{3.16}$$

Since the operating window is adjusted after T, the above step change can be done at most once every T time. This gives us a maximum safe discrete rate of decrease for the *averaged* $r(t)$:

$$\text{if} \qquad r_{av}(t) = \tfrac{1}{T}\int_{t-T}^{t} r(\tau)d\tau$$

$$\text{the condition} \quad -(r_{av}(t) - r_{av}(t-T)) < (1-g)r(t) + \tfrac{Q_{max}-X_0}{T} \tag{3.17}$$

will guarantee no cell loss. Note that when g is sufficiently small, $r(t)$ can jump to zero without causing any cell loss.

The drawback of a small g is that the throughput can be diminished. Specifically, if $r(t) > \frac{X_0}{T(1-g)}$, the queue will be empty and the link under-utilized. The throughput of the channel in steady state is

$$throughput = u(t) = \min\left[r(t), gr(t) + \frac{X_0}{T}\right] . \tag{3.18}$$

Choosing the appropriate g depends on the characteristics and functionality of the system. A low g will achieve a higher guarantee that cells are not lost. A higher g will achieve a higher guarantee that the service rate will be matched. If $r(t)$ is on average low ($r(t) < Q_{max}/T$) or $r(t)$ varies too rapidly ($\dot{r}(t) > Q_{max}/T^2$), then a low g might be preferable. If $r(t)$ is on average high or varies slowly, then a high g is more efficient.

3.2.5 Non-Linear Control with RRC and QRC

For most data traffic applications, achieving the maximum throughput is more important than strictly guaranteeing no cell loss. Since packets can be retransmitted at the transport layer, a faster cell throughput with some losses is more efficient than a loss free transmission with a limited throughput. Furthermore, it might be possible to control the behavior of the switch so as to limit the rate of change of $r(t)$ in order to prevent cell loss. Therefore, for the "average" application we should ensure that the rate command $u(t)$ can always follow service rate $r(t)$.

From the previous subsection, we know that g should be low when $r(t)$ is low, and g should be one when $r(t)$ is high. A $g = 0$ at low rates guarantees cell safety, while $g = 1$ at high rates guarantees maximum throughput. To have both g's at different time, we can construct a non-linear, although piece-wise linear, system G as in the original figure 3-2. The output of G is:

$$G(r(t)) = \max\left[r(t) - \frac{X_0}{T}, 0\right] . \tag{3.19}$$

If X_0 is set to Q_{max}, $G(r(t))$ will have zero slope in the interval $0 < r(t) < \frac{Q_{max}}{T}$ and a slope of one from then onwards. In the first interval, the system will be completely

cell loss free and will maximize the buffer to utilize future bursts in r(t). In the second interval, the system continues to maximize throughput and has a steady-state queue size of zero to minimize the probability of cell loss.

3.3 Dual Control Model

At the beginning of this chapter, we proposed a general model for a single node data connection. In section 3.1, we narrowed our model to the case where the effective queue estimator in the QRC uses a Smith Predictor. In this section, we want to analyze an alternative case of the general model. In this section we are still assuming that the controller's gain k is large ($k >> 1/T$).

3.3.1 Using Accounting QRC

A Smith predictor is not the only way to obtain an estimate of the effective queue size. As discussed in chapter 2, this estimator could use an accounting system. The accounting system adds to a register all the cells that are requested from the switch and subtracts from the same register all the cells that are received at the switch. The register counts the cells that are "in flight." The sum of this register and the the real queue size gives the controller the size of the effective queue size.

If we were to use QRC only, replacing the Smith predictor with an accounting system will not change the behavior of the overall control. The advantage of this scheme is that it does not use any estimate of the round-trip time T. However, when we are using both RRC and QRC, replacing the Smith predictor in the QRC by an accounting system will cause the overall control to be unstable. In order to maintain the same overall control scheme, we need to change the RRC whenever we change the QRC.

We now change the subsystem G in the RRC so that it is an LTI system with transfer function:

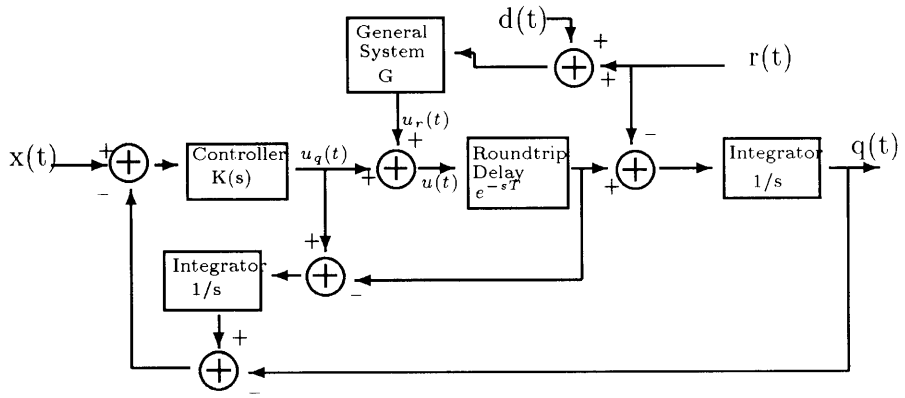

Figure 3-9: Dual Control Model

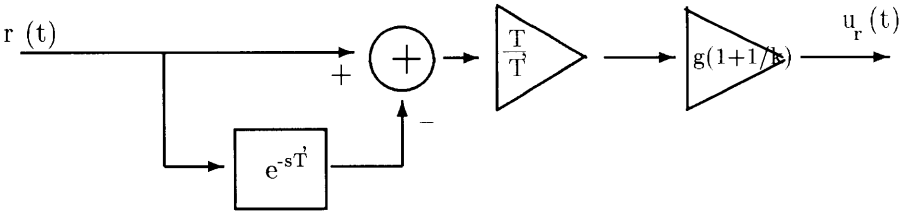

Figure 3-10: New G Subsystem

$$G(s) = \left(g + \frac{1}{k}\right)(1 - e^{-sT}) \ , \tag{3.20}$$

or equivalently a time domain description:

$$G_{out}(t) = \left(g + \frac{1}{k}\right)[G_{in}(t) - G_{in}(t - T)] \ . \tag{3.21}$$

The new system with the modified QRC (accounting subsystem in place of the Smith predictor) and the modified RRC (using the G in equation (3.20)) has almost the same behavior as our original system. We will call the new system the *dual* QRC+RRC system as opposed to the *primal* QRC+RRC system that we have discussed until now. The dual QRC+RRC system and its subsystem G are shown in figures 3-9 and 3-10 respectively. For the moment, we can assume that in figure 3-10, $T' = T$.

The dual QRC+RRC system has the same dynamics as the primal system. All

the observations made in section 3.1 still hold. The transfer functions between the inputs $r(t)$, $x(t)$, $d(t)$ and the output $q(t)$ are the same. Also note that the gain g has the same function in the dual structure as in the primal structure. It can be diminished to make the system safer, and it can be made piece-wise linear.

3.3.2 Primal Versus Dual Control Structure

Even though the input/output relations of the primal and dual control systems are equivalent, the structure and the inner functioning of the two systems are very different.

In the primal structure, we observed that the RRC was responsible for requesting all the data traffic during steady-state operation. The QRC was responsible for correcting the error in the RRC during transients. On the other hand, in the dual structure, it is the other way around. The QRC requests all of the data traffic during steady state, while the RRC corrects QRC's actions during transients. The dual structure works in a similar way as a credit based scheme, where the number of credits change dynamically as the service rate changes.

In the original structure, since the QRC had to correct the RRC, the QRC had to predict the behavior of the network. Therefore the QRC used an artificial delay which simulated the real round-trip delay in the network. On the other hand, in the dual structure, it is the RRC which predicts the behavior of the network, and it is the RRC which incorporates the simulated delay. By shifting from the primal to the dual structure, we are moving the burden of simulating the round-trip delay from the QRC to the RRC.

For the primal structure, the simulated delay in the QRC must be the same as the real round-trip delay in the network. As shown in [14], a difference between the real and simulated delays can cause the queue size to oscillate and even to become unstable. Nevertheless, as long as the error is within certain limits, the queue size can be guaranteed to be stable, and the oscillations will be centered around the desired queue size. On the other hand, for the dual structure, an error in the round-trip time estimate will not cause any oscillations or create any instabilities. A timing error will

only cause a steady-state error in the queue size. If T is the real round-trip delay in the network and $\hat{T}$ is its estimate used in the artificial delay, the steady-state error in the queue is

$$\Delta q(t) = r(t)[\hat{T} - T] \ . \tag{3.22}$$

In applications where the service rate $r(t)$ is low ($r(t) < Q_{max}/T$), the dual structure of the QRC+RRC control is preferable because it guarantees no oscillations and instabilities at the cost of a *small* (proportional to $r(t)$) offset in the queue size. In other applications where $r(t)$ is large, the queue size offset might be unacceptably large, and the primal QRC+RRC structure is preferable because it eliminates this offset at the cost of risking some oscillations and instabilities.

A potential problem with the dual QRC+RRC structure is that the estimate of the effective queue will accumulate an error when cells are lost during the connection. When the link is lossy, this problem can be corrected by occasionally sending special control cells which compare the total number of cells that have been requested by the switch with the total number that have been received. The discrepancy is then used to correct the effective queue size estimate.

3.3.3 Varying the Dual RRC

The introduction of the artificial delay in the dual RRC, opens the way for new variations that were not possible using the primal control structure. Specifically we can vary the parameter T', as suggested in figure 3-10, to create a different subsystem G. The transfer function for the new G will be

$$G(s) = \frac{T}{T'}\left(g + \frac{1}{k}\right)\left(1 - e^{-sT'}\right) \ . \tag{3.23}$$

This new G will cause the entire control system to have a variable impulse response

h_r, depending on the parameter T'. As shown in figure 3-11, a larger T' parameter will cause the positive part of the impulse response to stretch, while a smaller T' will cause it to contract. Since the total area of the impulse response is zero, the steady-state error of all these systems is zero for any T'. A smaller T' corresponds to a faster and more energetic response of the system to $r(t)$ variations. A larger T' corresponds to a slower and softer response. Note that with a very small T' we approach the limit established in theorem 1 and obtain the shortest possible transient response time.

The first part of the impulse response is unchangeable since it corresponds to the time when the system is starting to react to $r(t)$ variations. Because of the round-trip delay of T, it takes at least this amount of time for the results of the initial reaction to take place. However, the second part of the impulse response ($t \geq T$) can be shaped in any way using the dual control structure. By superimposing a weighted sum of delayed $G(s)$, we can set the second part of the dual impulse response to be *any* arbitrary function.

3.3.4 Optimal Control

We have shown that by using the dual RRC+QRC model, we can design a queue size controller with *any* impulse response we desire (excluding the first the region $0 \leq t < T$ which is unchangeable). The problem we face now is deciding which impulse response is optimal for some particular application.

Invariably, there is a tradeoff between the speed of the controller's response and its "stableness." A short impulse response will make the control system react very rapidly but will make the system over-sensitive and react very abruptly to minor changes over small amounts of time. A longer impulse response will filter and soften these abrupt responses at the cost of slowing down the whole system.

In general, we can say that we would like to obtain a system that could minimize the following criteria:

- The settling time or time required for the system to regain steady-state operation after a change in the available rate.

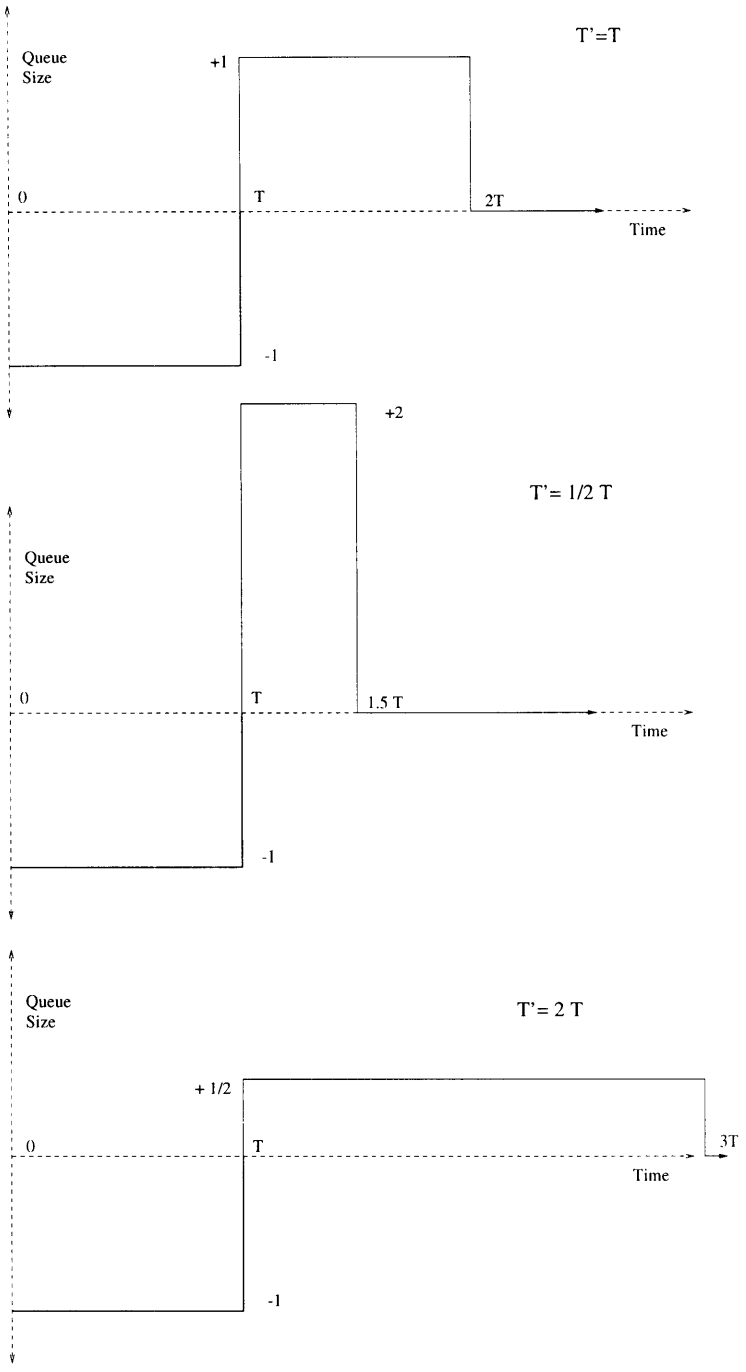

Figure 3-11: Some Possible Impulse Responses Obtained by Varying $G(s)$

- The variance of the queue size $q(t)$. This is an indication of the probability that the queue size will be outside a certain band around the reference level.

For the second criterion, we will assume that $r(t)$ consists of white noise with a constant intensity. The white noise approximation also applies as long as the decorrelation time is less than the round-trip delay T. With this assumption, the variance of $q(t)$ can be obtained by using probabilistic linear system theory:

$$\begin{aligned}\sigma_q^2 = K_{qq}(0) &= [\hat{K}_{hh} * K_{rr}](0) \\ &= [\hat{K}_{hh} * \delta(\tau)](0) \\ &= \hat{K}_{hh}(0) \\ &= Energy(h(t)) = \int_0^\infty |h_r(t)|^2 dt \ , \end{aligned} \tag{3.24}$$

where $\hat{K}_{hh}$ is the deterministic autocorrelation of our system, and K_{qq} and K_{rr} are the probabilistic autocorrelations of the queue $q(t)$ and the rate $r(t)$ respectively.

We can now pose our problem as:

minimize:

$$\sigma_q^2 \times \text{settling time}$$
$$or\ Energy[h(t)] \times (T + T')$$

constrained by:

$$Expectation[q(t)] = X_0 \ \ (\text{reference level}) \ . \tag{3.25}$$

Using Cauchy-Schwartz inequality and calculus, we can show that the minimization takes place when the impulse response is anti-symmetric. This is shown in the first impulse response of figure 3-11. The impulse response takes the value of one from T to $2T$. This impulse response can be synthesized on a dual RRC+QRC controller

by using a single $G(s)$ with $T' = T$ and $g = 1$. This impulse response is also the one obtained from the primal RRC+QRC controller when $g = 1$.

Even though the control system with $T' = T$ is "optimal" according to the above criteria, we might still wish to have a different T' for some particular scenarios. In the case where the available rate $r(t)$ is very bursty, we might want to make T' larger to filter out the high frequency components. If $r(t)$ is mostly steady with a few occasional steps, we might want T' to be smaller to react faster to those step changes.

Chapter 4

Implementation

In chapters 2 and 3 we have studied the problem of flow control without considering its implementation in a real network. Our models in chapter 3 were idealized single node representations of a network system. We assumed a fluid data approximation, a continuous feedback, and no computational effort to obtain information such as the available switch rate. We also made no efforts in extending the model to the multi-node scenario.

This chapter will show how the results in chapter 3 are still valid when the simplifying assumptions do not hold. Section 4.1 shows how to build the QRC and RRC controller on a real single node network. The issues discussed are how to implement an effective discrete feedback and how to estimate the available rate when more than one connection is using the node. Section 4.2 explains how to extend the control scheme to the multi-node case. Finally, in section 4.3 simulation results are presented which corroborate the analytical predictions.

Since ATM's Available Bit Rate service (ABR) offers the type of tools that we need for an explicit flow control scheme, the control implementation will be described within the ATM framework. In ATM, data travels in regular cells, while control information travels in special resource management (RM) cells. As in our model, the switches calculate the desired source explicit rate (ER) and send their command inside RM cell traveling toward the source.

4.1 Single Node

4.1.1 QRC Feedback Using Sample and Hold

In ATM circuits, feedback information can be transported by backward RM cells returning to the source. Since RM cells arrive at discrete time intervals, the control scheme must be modified to work with discrete feedback.

A straightforward way to perform discrete feedback is to send to the source a sample of the command $u_q(t)$ every time a backward RM cell passes by the switch. When the source receives a feedback RM cell, it sets its sending rate to the value of the command sample $u_q[n]$. The source then continues sending data at that same rate as long as no new RM cells arrive. When the source does receive a new RM cell, it will set the sending rate to the new value.

The discretization of the feedback will cause some degradation of the performance, and if the RM cell frequency is too low the system might become unstable. In our QRC system, the time constant is the inverse of the gain or $1/k$. According to the Nyquist criterion, as long as the inter-arrival time between RM cells Δt_{rm} is less than half of the time constant, the system will be stable:

$$\Delta t_{rm} < \frac{1}{2k} \, . \tag{4.1}$$

This stability requirement limits the value of k. This means that if the frequency of the RM cells is small, k must take a small value. Remember that a large value of k results in a better controller. Furthermore, even if the system is stable, but Δt_{rm} is close to the Nyquist limit, the system will exhibit unwanted oscillations during transients and will take longer to stabilize than our continuous approximation.

4.1.2 QRC Feedback Using Expiration

A more natural approach to discretize the feedback takes advantage of the meaning of the information carried in a sample of $u_q(t)$. When the switch sends a discrete

command $u_q[n]$ to the source, it is actually saying that it needs $u_q[n]/k$ cells to fill the effective queue up to the reference level. The source can accomplish this by sending data at a rate of $u_q[n]$ during $1/k$ time. After $1/k$ time, the rate $u_q[n]$ is no longer valid,and the source shuts off. If a new RM cell arrives at the source at any time, its command will be the new valid rate for $1/k$ time. If t_{rm} is the time when the last RM cell arrived, the command of this RM cell $u_q[n]$ is the current source rate until $t_{rm} + 1/k$ or another RM cell arrives, whichever comes first.

The algorithm operation at the source and switch can be summarized as follows:

at the switch:

$$q^\star(t) = q(t) + \min(t - t_{rm}, 1/k) \times k[x(t_{rm}) - q^\star(t_{rm})] \qquad (4.2)$$
$$u_q[n] = u_q(t_{rm})$$

at the source:

$$QRC\ s(t) = \begin{cases} u_q[n] & \text{if } (t - t_{rm}) < 1/k \\ 0 & \text{if } (t - t_{rm}) \geq 1/k\ . \end{cases} \qquad (4.3)$$

This feedback scheme enables the controller gain k to be as large as we want. Since the switch will never request more cells than would fill the reference level, the system is stable for any k.

4.1.3 QRC Integrated Credit-Rate Feedback

Even though the previous QRC feedback guarantees stability, it does not specify a value for the gain k. When the RM cell inter-arrival time Δt_{rm} is large it makes little sense to have a large k since the response will be slow anyways. Having a $k >> 1/\Delta t_{rm}$ will only provide marginal increases in the system's response velocity. Moreover, a large k can create bursty data flows in the incoming links, which can disturb other connections traveling through those links. Therefore, we would like our

feedback system to incorporate a dynamic k which can vary according to Δt_{rm}.

We would like the expiration time $1/k$ to be equal to or proportional to the RM cell inter-arrival time Δt_{rm} during which that gain is valid. Since we don't know what the next interval Δt_{rm} will be, we can use the last interval as an estimate. When a new RM cell arrives at time t, this estimate is computed. We will denote the estimate as $\Pi_{rm}(t) = t - t_{rm}$, where t_{rm} is the time when the last RM cell passed by. The current k then becomes:

$$k(t) = \frac{1}{\alpha \Pi_{rm}(t)} = \frac{1}{\alpha(t - t_{rm})} , \tag{4.4}$$

where α is the proportionality constant.

When $\alpha >> 1$, the feedback samples are expected to expire after the arrival of the next RM cell. Therefore the system is not responding as fast as the feedback rate enables it. When $\alpha < 1$, the feedback samples are expected to expire before the arrival of the next RM cell. Therefore the source will not send any data for some period of time. This effect will create bursty rates in the incoming cells. A good value for this constant is $1 < \alpha < 10$.

The switch needs to inform the source of the expiration time $1/k$. Therefore the backward RM cell must contain not only the sample command rate $u_q[n]$ but also the expiration time τ_{exp} of that sample. Note that sending *both* rate and time information in the RM cell is equivalent to sending joint credit and rate information. From a rate perspective, the switch tells the source to send data at a given rate for a certain amount of time. From a credit perspective, the switch tells the source to send a certain amount of cells at a given rate.

This integrated credit-rate method of sending feedback is an improvement over the purely credit or rate mechanisms. With pure credit feedback, the source does not know at what rate to send the requested cells. With pure rate feedback, the source does not know for how long the given rate is valid if no further RM cells arrive. By incorporating *both* rate and time in the feedback information, we provide

extra information and improve the performance of the two feedback schemes working independently. Only when the feedback rate goes to infinity does the rate-only feedback perform as well as the integrated credit-rate feedback. This is the continuous feedback case discussed in the previous chapter.

4.1.4 RRC Feedback

The feedback for the RRC control is straightforward and follows from the QRC Feedback. The switch estimates the service rate $r(t)$ through one of the algorithms described in the next subsection. The estimate $\hat{r}$ is sent to the source as well as an expiration time t_{exp}. When the source receives the RM cell, it will send data at rate $\hat{r}$, and if no RM cell is received before, it will shutdown at t_{exp}. This expiration time should be sufficiently large so that rate shutdown in RRC is a rare event which only happens in the extreme cases where feedback is interrupted.

4.1.5 Determining the Service Rate

We would now like to extend our model to the case where there are many users being served by a single switch. We would like the rate allowed for each VC to be the fair rate in the max-min sense. This means that any one particular VC cannot increase its share of the bandwidth by decreasing the share of another VC with a lower rate. If a_i is the arrival rate for channel i and C is the total bandwidth of the switch, the fair rate for channel j is the correct f_j such that the following equation is satisfied:

$$\sum_{i \neq j} \min(a_i, f_j) + f_j = C \ . \tag{4.5}$$

If we assume that the switches use per-VC queuing for each of the users, we can guarantee that a particular channel will not send data across a switch at a rate faster than its fair rate. Furthermore, with per-VC queuing, the fair rate for any channel is the same as its *available* rate on that switch. The available rate is the *maximum* rate

at which that VC can have its cells serviced by the switch. The available rate is also the maximum rate at which a source can send data without increasing its per-VC queue at the switch.

We can also use the per-VC queue as a computational device to estimate the fair or available rate for each VC passing through the switch. In this case, the available rate of any VC is the hypothetical service rate for when the queue of that particular VC is never empty. If this were the case, the switch would service that VC exactly once in every cycle of the round-robin. If in time T, the switch services x_i cells for each channel i (with time Δ needed to service each cell) and cycles N times around the round-robin, the available cell rate $\hat{r}_j$ for channel j can be computed with the following iteration:

$$T_j^0 = [\sum_i x_i]\Delta \qquad x_i^0 = x_i \tag{4.6}$$

$$T_j^1 = [\sum_{i \neq j} x_i^0 + N]\Delta \qquad x_i^1 = \min[N, \frac{T_j^1}{T_j^0} x_i^0] \tag{4.7}$$

$$\vdots \qquad\qquad \vdots$$

$$T_j^n = [\sum_{i \neq j} x_i^{n-1} + N]\Delta \qquad x_i^n = \min[N, \frac{T_j^n}{T_j^0} x_i^0] \tag{4.8}$$

$$\hat{r}_j = \frac{N}{T_j^n} \, . \tag{4.9}$$

It can be shown that this iteration converges. The result of the iteration gives us the rate that a particular VC would use *if* its queue were always nonempty and the other VCs were to continue transmitting data at the same rate. This is equivalent to the available bandwidth that we defined above.

The calculation in equation (4.8) requires the switch to count at every interval T all the cells that it services for each VC. Furthermore, the switch might have to perform a theoretically infinite number of calculations to determine $\hat{r}_j$ for each

different channel. This exact calculation can be simplified considerably by using the first or zeroth order iteration only. When the number of VCs is large, the total real number of cells serviced and the hypothetical number of cells serviced for each VC is approximately equal. This means that $\sum_{i \neq j} x_i + N \approx \sum_i x_i$, making the zeroth order approximation (4.6) quite accurate. This calculation only requires the switch to keep count of the number of round-robin cycles that it is performing for every interval of time. We call the zeroth order approximation $\hat{r}_{max}$.

Since $\hat{r}_{max} \geq \hat{r}_j$, this approximative algorithm will tend to overestimate the available rate for channels with very low initial data rates. If a source j starts sending data at this overestimated rate, after some time, the queue for channel j will be nonempty. The increase in traffic will decrease $\hat{r}_{max}$. Specifically, with a non-empty queue j , $x_j = N$ and $\hat{r}_j = \hat{r}_{max}$. The new lower $\hat{r}_{max}$ will give channel j its exact available rate. Therefore, the rate excess caused by the error in the $\hat{r}_{max}$ approximation will only last for one round-trip delay.

Even though the $\hat{r}_{max}$ approximation can cause some rate overshoot for one round-trip delay, this overshoot is negligible when there are a large number of virtual channels. For example, let us suppose that there are 100 VCs passing through one switch, 99 VCs are sending data at the full capacity, and one VC is not sending data. The 99 VCs are receiving the exact available rate ($\hat{r}_{max} = \hat{r}_j$), and the other VC is receiving a rate less than 1/100 higher than the exact value ($\hat{r}_{max} = 1.0099\hat{r}_j$). Since this small error will only persist for one round-trip delay, the effects of the approximation error are negligible. Furthermore, the small error can be corrected by the QRC algorithm. In the simulations presented in section 5.4, we only use the first order iteration (4.7) to calculate the available rate and did not notice any loss in performance.

The zeroth order approximation described in equation (4.6) uses the per-VC queues to perform an algorithm very similar to Anna Charny's algorithm to determine the max-min fair rate [5].

4.2 Multiple Node Extension

In this section we will extend our primal control model (figure 3-2) to the multi-node case. We continue using per-VC queuing at each switch.

4.2.1 Primal Control Extension

In the multi-node case, there is more than one switch between the data source and the destination for some virtual channel. Using the primal controller at each switch, we can obtain separate information for the QRC and RRC cell requests $u_q^i(t)$ and $u_r^i(t)$ at each switch i. The sum $u^i(t) = u_q^i(t) + u_r^i(t)$ is the rate needed to stabilize the VC queue at switch i. Because the source cannot accommodate the data rate demands of all the switches in a VC, it needs to set its sending rate to the minimum of these requests:

$$\text{data rate } = \min\left[u_q^i(t) + u_r^i(t)\right], \forall i \ . \tag{4.10}$$

Only the node with the lowest cell request will be able to stabilize its queue at its reference level. The queues for the same virtual channel at other nodes will either be empty or emptying out. The QRC+RRC control scheme described in the previous sections regulates the flow of data between the source and the bottleneck node. We will refer to this node as the *active* node.

The multi-node control can be implemented using the integrated credit-rate feedback discussed in subsection 4.1.3. Backward RM cells carry three fields of information:

- Requested RRC rate: u_r
- Requested QRC rate: u_q
- RM inter-arrival time: $Time_{rm}$

Periodically, the receiver sends a backward RM cell upstream towards the source. Originally, the first two fields are empty while the third field contains the time when the next backward RM cell is expected to be sent. At each node i , whenever an RM cell is received, it calculates its requested rates u_q^i and u_r^i using the gain $k = 1/\alpha Time_{rm}$. If the calculated u_r^i is less than the u_r field in the RM cell, u_r is replaced with u_r^i. Then, the u_q field is set to $\min\left[\left(u_r^i + u_q^i\right), (u_r + u_q)\right] - \min\left[u_r, u_r^i\right]$. This formula forces both the new RRC and new sum of QRC+RRC commands to be each the lowest of all the switches. When the backward RM cell arrives at the source, it will contain the lowest u_r and the lowest sum of u_r and u_q of all the switches. In steady state, both commands are determined by the bottlenecked or active node. However, during transients, the active node (with the minimum $u_q + u_r$) does not necessarily have to be the one with the lowest RRC command, u_r.

Note that if the expiration time for the RRC command is the same as that for the QRC command, $Time_{rm}$, we can collapse the RRC and QRC rate fields (u_r, u_q) into a total rate field. The source needs only to send data at this total rate without worrying about its individual components.

4.2.2 Max-Min Fairness

Now that we have implemented the QRC+RRC control algorithm for the multi-node case, we would like to make some observations on its global performance. This algorithm establishes a feedback mechanism between the source and the active node for each VC. The dynamics of the VC queue at the active node are the same as those for the single node case which we have analyzed in all the preceding sections. As in the single node case, the multi-node algorithm will stabilize the queue level of its active node. In steady state, if $g = 1$ (we are fully superposing QRC and RRC) the active node will have a queue size equal to the reference level. The other nodes in that VC will have empty queues. We can now make the following statement on the global behavior.

Theorem 3 The full superposition of the primal QRC and RRC algorithms in the multi-node case provides a max-min fair allocation of bandwidth among the different virtual channels.

Proof: In steady state, each virtual channel will include at least one node (the active node) where the queue size is equal to a reference level. Since the service discipline is round-robin, the active node of a particular VC will service that VC no slower than all the other VC's passing through that node. Therefore, the link following the active node of some VC is a bottleneck link for that VC. Since every VC has a bottleneck link, the overall rate allocation is max-min fair. □

Note that if $g < 1$ and we are only using a weighted superposition of RRC and QRC (or QRC only) the allocation might not be max-min fair. This is because at sufficient high rates, the active node will have an empty queue. The link following the active node will be underutilized and will cease to be a bottleneck link. With some VC not having a bottleneck link, the overall allocation cannot be max-min fair.

4.3 Simulations

In this section we present some of the simulation results using the primal QRC+RRC flow control algorithm. The algorithm is first simulated in the single node case using Matlab's Simulink package. The multi-node case is simulated using Opnet. In all cases, we assume that the sources are greedy and always have data to send.

4.3.1 Single Node Case

The first scenario that we simulate consists of a single node network. The round-trip delay between the source and the switch is 40 milliseconds.

The primal RRC+QRC control algorithm fully utilizes the RRC ($g = 1$). The switch is operating at rates between 40,000 and 100,000 cells/sec. Control information is sent to the source with a constant frequency of 1/64 the maximum data rate, or

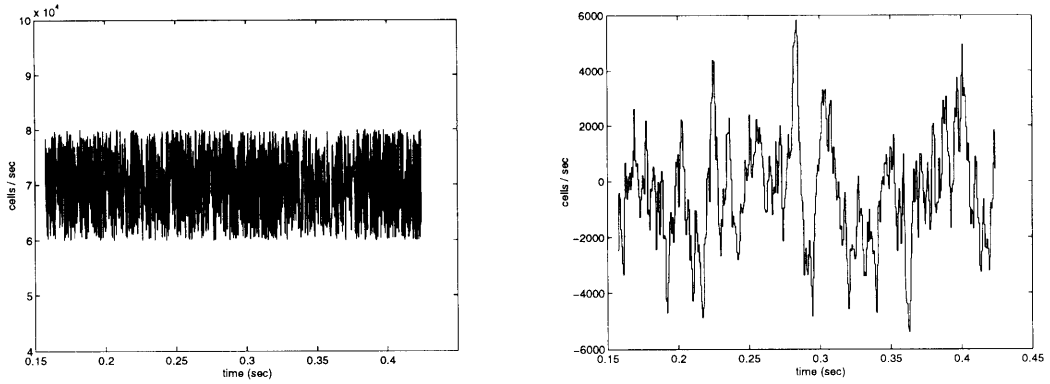

Figure 4-1: RRC and QRC Commands with a Band-Limited White Noise Service Rate

1562.5 Hz. The QRC gain k is set to $0.2/t_{rm}$ or $312.5s^{-1}$.

The single node case was simulated extensively using the Simulink package. The single node network and controller were modeled using adders, delays, gains, integrators, and zero-order holds. The model is equivalent to the continuous time system presented in figure 3-2, with two exceptions. First, the feedback is discretized by a zero-order hold which only releases new feedback information every RM cell period (0.64 msec in our example). Second, since the total rate requested cannot be negative, the minimum QRC command is limited to the negative of the RRC command.

Random Service Rate

In the first simulation, the service rate is band-limited white noise. At every sampling time in the simulation, the service rate is a random value between 60,000 and 80,000 cells/sec. On the left side of figure 4-1, one can observe the oscillations of the service rate. The RRC command shown on that figure follows the actual service rate. In the right side of the same figure, we can observe the QRC command for that scenario.

In figure 4-2, we can observe how the real queue size changes in response to the service rate and our control algorithm. The desired queue size is set to 50 cells. Notice that for service rate changes of tens of thousands of cells/sec, the queue size changes a few tens of cells. The small impact of the service rate on the queue size is in accordance with our theory, which states that the oscillations (response) of the queue

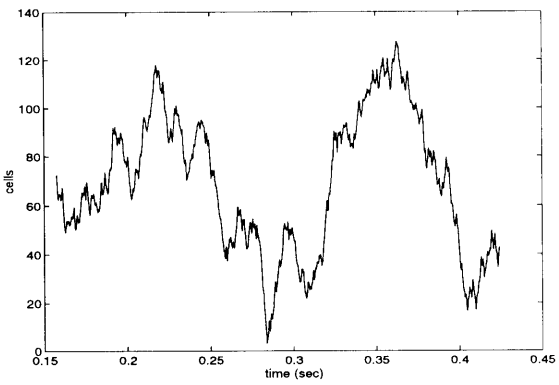

Figure 4-2: Queue Size with a Band-Limited White Noise Service Rate

size become smaller as the frequency of the service rate increases. The algorithm is also shown to be stable, even for highly varying service rates like this one.

Step Changes in Service Rate

In our second simulation, the random service rate is replaced by a service rate with step changes. The service rate starts at 70,000 cells/sec and jumps to 45,000 cells/sec at 0.1 seconds. The rate jumps back to 70,000 cells/sec at 0.25 seconds. On the left side of figure 4-3, we can observe the RRC command as it follows the service rate. On the right side of the figure, we can see the corresponding QRC command.

Since the round-trip delay is 40 msec, the RRC command will be 40 msec late. We can see that during the 40 msec between 0.1 and 0.14 sec, the QRC is commanding a rate of -25,000 cell/sec (the change in rate of the step) that compensates the error due to the RRC command's lateness. The negative QRC rate is subtracted from the RRC command. Again at time 0.25 sec, the QRC commands a rate to refill the empty queue caused by the latency of the RRC command.

In figure 4-4, we can observe how the queue size varies in time. Between time 0.1 and 0.14 sec, the queue size increases steadily due to the decrease in service rate. At time 0.14 sec, the command from time 0.1 sec starts taking effect and steadily decreases the queue size until it reaches the desired level at about 0.18 sec. This in accordance with the theory established in sections 3.1 and 3.2. As figure 3-4 predicts,

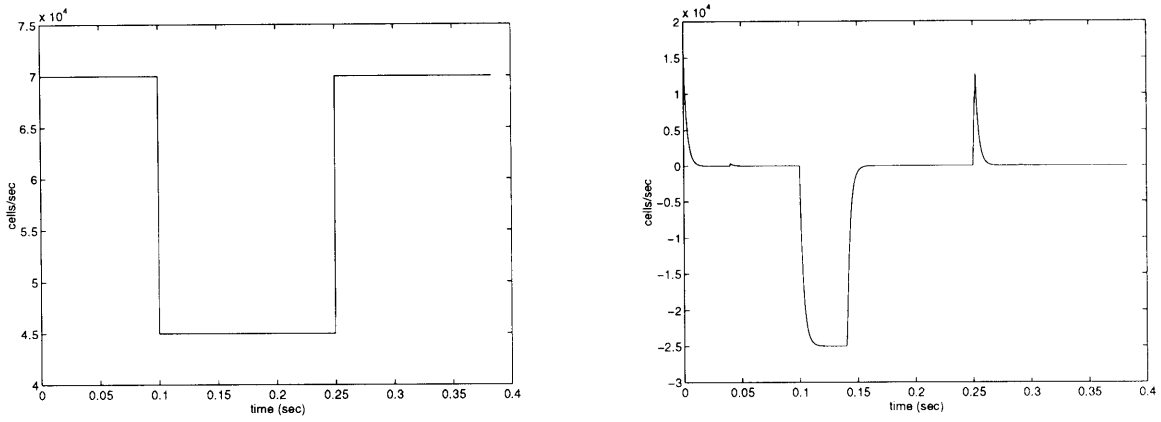

Figure 4-3: RRC and QRC Commands with Step Changes in Service Rate

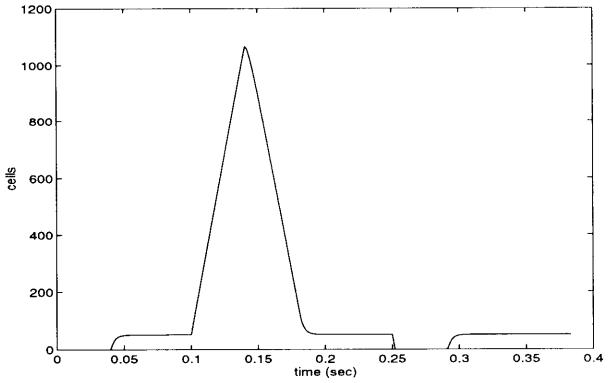

Figure 4-4: Queue Size with Step Changes in Service Rate

the queue size changes linearly for one round-trip time and then is corrected in the next round-trip time.

At time 0.25 sec, the step change in the service rate causes the queue to empty out. The QRC controller does not respond by commanding 25,000 cells/sec for one round-trip delay but commands just enough cells to fill up the queue to the desired level at 0.29 sec. When the queue size saturates (the queue empties out), RRC's error is less than the step change in rate times a round-trip delay. Since the QRC controller is an observer of RRC's error, the QRC knows this and decreases its command accordingly.

4.3.2 Multiple Node Case

In the multiple node scenario, we simulated the algorithm using Opnet. This simulation uses the multi-node extension of the algorithm, which is discussed in section 4.2. Each switch estimates its service rate according to the first order approximation presented in subsection 4.1.5. The estimation of the RRC rate is performed every 40 cycles of the per-VC queue round-robin, enough to ensure that the error is only about 2%. The backward RM cells are routed out-band (that is, using separate channels). Each switch implements a per-VC queue servicing discipline at every output port.

The network configuration that we are simulating is depicted in figure 4-5. Sources 0, 1, 2, and 3 are sending data to receivers 3, 0, 1, and 2 respectively. The propagation delays between the switches is 10 msec, and that between the switches and users is 0.5 usec. Switches 0, 1, 2,and 3 can transmit data at each of their ports at rates 7000, 8000, 9000, and 10,000 cells/sec respectively. Like in the single node case, the RM cell frequency is set to 1562.5 Hz, and the QRC gain is set to $1/(5t_{rm})$ or 312.5 s^{-1}. The desired queue level is 50 cells for each of the VC queues.

At initialization, none of the sources are sending any data. The receivers are continuously sending RM cells to the sources at a fixed frequency. At time 0.05 sec, source 0 starts sending data to receiver 3 on VC 3. Because switch 0 is the slowest node, it becomes the bottlenecked and active node. At time 0.15 sec, source 3 starts sending data to receiver 2 on VC 2. Because switch 2 has to transmit cells belonging to VC's 2 and 3, it becomes the next active node. At time 0.3 sec, source 2 starts sending data to receiver 1 on VC 1. This causes switch 1 to become the next active node. Finally, at time 0.4 sec, source 1 starts sending data to receiver 0 on VC 0. Switch 0 becomes once again the active node.

In figure 4-6, we can observe the RRC commands that the switches compute for the sources. The data samples are recorded at the time when the RM cell with the command arrives at the corresponding source. Initially, the RRC command for source 0 is 70,000 cell/sec, which is the rate of the slowest link in the path. When source 3 starts sending data, the RRC command for both sources becomes 45,000

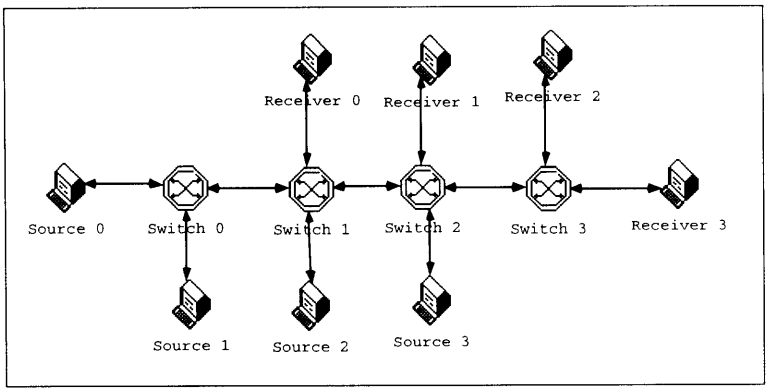

Figure 4-5: Multiple Node Opnet Simulation Configuration

cells/sec, which is half the rate of the node they share (switch 2). When source 2 starts transmission, the RRC commands for sources 0 and 2 become 40,000 cells/sec, since that is half the capacity of the node they share (switch 1). Since source 0 is only using 4/9 of switch 2's capacity, the RRC command for source 2 is increased to 5/9 of switch 2's capacity or 50,000 cells/sec. Finally, when source 1 starts sending data, the RRC command for sources 0 and 1 becomes 35,000 cell/sec (half of switch 0's capacity), and sources 2 and 3 increase their rates to 45,000 and 55,000 cells/sec respectively to utilize the unused bandwidth in switches 1 and 2.

The RRC commands shown in figure 4-6 are max-min fair to all the sources. However, we can also see that it takes some time (delay between switch and source) for a new RRC command computed at the switch to be followed by the source. For example, at time 0.15 sec, switch 2 send a lower RRC command to source 0. However, the command reaches source 0 at time 0.17 sec. Between times 0.15 and 0.19 sec, switch 2 is receiving from sources 0 and 2 more cells that it can service. This error cannot be corrected by the RRC controller since latency is an inherent characteristic of the network. However, this error can be corrected at a later time by the QRC controller.

In figure 4-7, we can observe the QRC commands that arrive at each source. Initially, when each source starts sending data, the initial QRC command is high in order for the source to send an initial burst of cells that will fill up the VC queue of

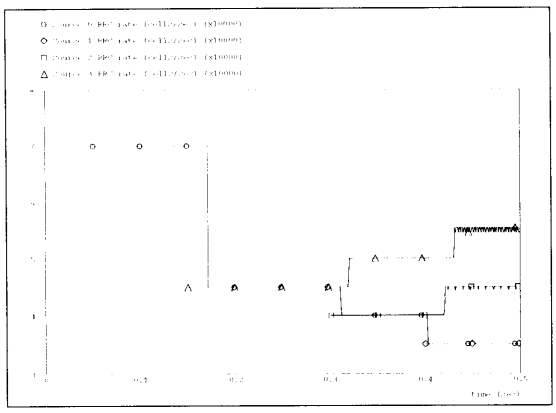

Figure 4-6: RRC Commands at Each Source

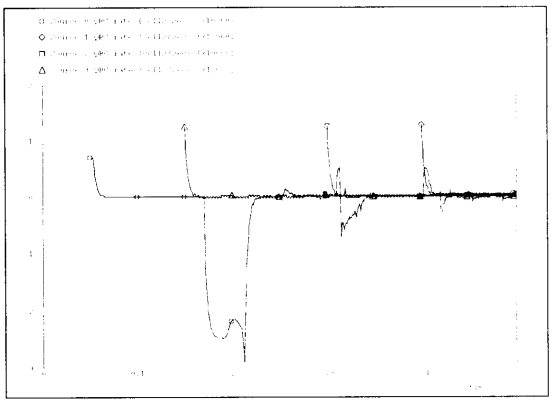

Figure 4-7: QRC Commands at Each Source

the active node to the desired size (50 cells in this case). After the initial burst, the QRC source quickly settles down to close to zero.

At time 0.17 sec the QRC command for source 0 goes negative in order to slow source 1 down. This is needed because source 3 has started sending data at time 0.15 sec and switch 2 has become congested. Since the QRC controller in switch 2 can observe the error caused by the delay, it can request the exact QRC rate needed to correct this error. Unlike the single node simulation, where the QRC command was a negative step, the QRC command between times 0.17 and 0.21 sec is slightly deformed due to the extra cells that arrive from switch 0's queue and the fact that previous QRC commands of switch 2 were not satisfied (it was not the active node). However, the total area under source 0's QRC command is the same and it's effect is identical to the single node case. Likewise, at time 0.31 sec, the QRC command for Source 0 goes negative to reduce the congestion at switch 1 caused by the activation of source 1. Throughout the simulation, the small QRC commands correct the small errors in the RRC due to the errors in the estimation of the actual service rate.

In figure 4-8, we can observe the queues for VC 3 (the VC between source 0 and receiver 3) at each of the switches. At time 0.05 sec, the queue of the active node (switch 0) fills up to the desired queue size of 50 cells. Notice how the ramp is close to a first order exponential without any oscillations. At time 0.15 sec, when source 3 starts sending data, switch 2 becomes the active node and its queue starts increasing steadily for 0.04 sec (round-trip delay). Notice that this is the analogous situation

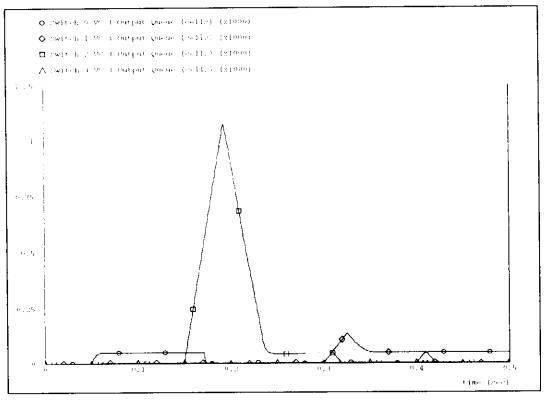

Figure 4-8: Output Queue of VC 3 at Each Switch

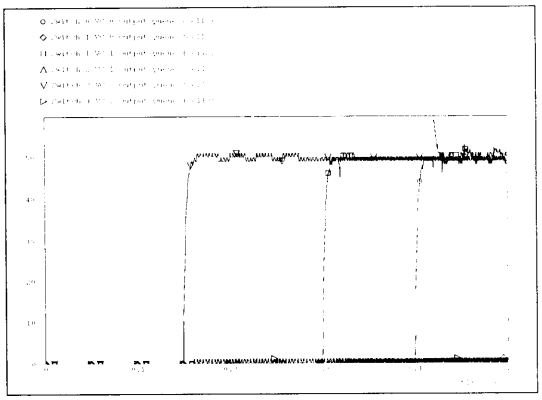

Figure 4-9: Output Queues of VCs 1, 2, and 3 at Each Switch

to that shown in figure 4-4 for the single node case. As the two figures show, the behavior of the queue of an active node in a multi-node network is very similar to that of the queue of a single node network.

At time 0.3 sec, switch 1 becomes the active node, and its queue increases for 0.2 sec (round-trip delay). Again, the QRC command forces the queue size to decrease to the desired level. At time 0.4 sec, switch 0 becomes the active node again. Since the delay to source 0 is negligible, there is no overshoot. Note that when a node ceases to be active, its queue empties out. At every moment of time between 0.05 and 0.5 sec there is at least one active node with a non-empty queue.

In figure 4-9, we can observe the VC queues for VCs 0,1, and 2. For each VC there is one queue in its path that is of size 50 (the desired queue size). Each queue fills up as soon as the corresponding VC starts transmitting data. For these VCs the queue sizes remain mostly stable throughout the simulation.

Chapter 5

Bi-Channel Rate Control

In this chapter, a new algorithm is presented that further decouples the QRC and RRC dynamics. While the algorithms in the previous chapters have separate (QRC and RRC) components at the controller, the feedback does not maintain the separation between the rate commands determined by the two components. Conceptually, the new algorithm uses the idea of virtual data to divide the feedback rate commands into two separate channels. To distinguish this new algorithm from its predecessors, we shall refer to it as bi-channel rate control (BRC). The separate channels allow for several improvements. First, BRC does not need an explicit estimate of the round-trip delay in its controller. Second, unlike its predecessors, the new controller is stable even under time-varying delays. Third, it stabilizes a queue in the shortest time possible given no knowledge of the round-trip delay.

The robustness of BRC against fluctuations in the network delays also allows for improvements in the queuing discipline of the switch. One such improvement is a mechanism in which all VCs can share a single FIFO queue instead of using per-VC queuing. While the real data is stored in a FIFO queue, the controller uses a *virtual* queue [6, 26, 27] to compute the rate and queue size estimates used in the RRC and QRC. The variations in the round-trip delay of one VC due to other VCs sharing the same FIFO queue will not cause any problems since the new algorithm can adapt to changes in the delay.

Section 5.1 describes the algorithm and shows how the independent QRC and RRC

feedbacks can be implemented. Section 5.2 explains and analyzes the performance of our algorithm assuming greedy sources. The algorithm is shown to be stable, max-min fair, and optimal in the sense that it minimizes the transient response time. Section 5.3 explains the concept of virtual queuing and shows how it can be used to implement this algorithm on switches which use FIFO queuing. Finally, section 5.4, simulates this algorithm on two different types of switches (one using per-VC queuing and the other using FIFO queuing) and show that the results confirm our theoretical predictions.

5.1 Algorithm

5.1.1 Concepts

The BRC algorithm uses two independent controllers that work in parallel and deal with different aspects of the flow control problem. The RRC algorithm matches the source rate with the available bandwidth of the VC. The QRC algorithm tries to stabilize the queue size of the slowest switch.

The BRC algorithm is an extension of the primal control structure presented in chapter 3. As in the primal case, the BRC algorithm relies on the RRC to perform the steady-state control while the QRC performs the transient control. However, unlike the primal (as well as dual) structure, the BRC algorithm does not require any knowledge of the round-trip delay.

As in the primal structure, the QRC algorithm can be seen as RRC's error observer. As an observer, the QRC needs to simulate the behavior of the real system. However, instead of using a Smith predictor to simulate a hypothetical data flow, the BRC uses information carried in the RM cells to simulate this flow.

This technique of using RM cells circulating through the network for simulation purposes will be referred to as using *virtual data*. This virtual data is information contained in RM cells which represents the amount of data that would flow in the network under hypothetical circumstances. By using virtual data, a single real net-

work can perform several virtual traffic simulations as well manage the real traffic. Because the BRC algorithm uses virtual data, it does not need an explicit knowledge of the round-trip delay. The addition of a virtual channel, gives rise to the name *bi-channel* rate algorithm.

5.1.2 Feedback Implementation

The first mechanism that must be established for our flow control algorithm is a feedback loop that carries information around the network. This is implemented by resource management (RM) cells which are circulated continuously between the source and the destination. While in the backward direction (from the destination to the source), the RM cells are routed out-band (i.e., using a separate channel), and we assume that their propagation delay is constant. Moreover, in the backward direction, the time interval between consecutive RM cells will be denoted as Δ_{rm}. This time interval does not have to be constant, but should be small enough to ensure a fast response of the system.

When the RM cell reaches the source, it is forwarded to the destination along the same VC path in a forward direction. This time, the RM cells are routed along with the data cells of the corresponding VC. Therefore, the delay of these forward RM cells is the same as the delay of regular data cells of the same VC.

In the backward direction, each RM cell contains four fields of information in which control data can be transferred between the various switches and the source within the same VC path. The first field is used exclusively by the RRC controllers and the other three are used exclusively by the QRC controllers. The four fields are:

- RRC rate: RRC_{rm}
- QRC rate: QRC_{rm}
- Time interval: $Time_{rm}$
- QRC request: Req_{rm}

The RRC and QRC rate fields contain the rate requests used by the independent RRC and QRC algorithms, respectively. The time interval field contains the time when the next RM cell is expected to arrive. This information is used by the QRC controller to calculate its gain. The QRC request field is used to keep track of how many cells each QRC controller has requested.

At the destination, the RM fields are initialized as follows: RRC_{rm} is set to the maximum rate the destination can receive; QRC_{rm} and Req_{rm} are set to zero; $Time_{rm}$ is set to the time interval between the current and previous RM cell's arrival times. The first RM cell can have any value in the $Time_{rm}$ field.

Every time that an RM cell passes through a switch, the switch obtains the information contained in that cell, performs some computation, and incorporates new information in the cell.

5.1.3 RRC

By using the method in subsection 4.1.5 (with whichever order of iteration we choose), the RRC can have an updated approximation of the available service rate of the switch for each VC. Every time a backward RM cell passes through the switch, the RRC will write into the cell's command field the *minimum* of the locally calculated rate and the rate contained in the RM field RRC_{rm}. When the backward RM cell reaches the source, the RRC command field will contain the minimum available rate of all the switches in the VC path.

By itself, the RRC command can ensure that the sources are sending data at the correct *average* rate. However, this scheme by itself is unstable and cannot control the queue sizes. If there is an average error in the measurement of the available rates at some switch, the queue at that switch might empty out or overflow. In order to stabilize the queue sizes and ensure a fair and smooth circulation of data, we need to introduce the QRC algorithm.

5.1.4 QRC

The QRC algorithm runs in parallel with the RRC algorithm. It uses feedback from measurements of the queue sizes to calculate its rate command. The QRC algorithm works independently from the RRC and corrects its errors. In this chapter, we carry the separation of the two algorithms into the feedback process. This allows us to design a QRC algorithm which does not assume any knowledge of the round-trip delay.

The key functionality of the QRC algorithm is to estimate the *effective* queue size of a VC. The effective queue size of a VC at a switch is the total number of cells in the VC's queue plus all the cells that have been previously requested by the QRC for that VC and are bound to arrive. The measurement of these cells that are "on the fly" requires communication between the various switches along the VC path. The method described below is the multi-node extension of the accounting scheme described in chapter 3.

Estimating the Effective Queue Size

The QRC in each switch maintains a register Q_{reg} indicating the *residual* queue size for each VC. The residual queue size consists of all the cells that are "on the fly" and have not entered the real queue yet. The sum of the real and residual queue sizes is the effective queue size.

Whenever a backward RM cell passes by the switch, the previous QRC command is multiplied by the minimum of $Time_{rm}$ and $k_0 \cdot Time_{old}$, where $Time_{rm}$ is the time interval field contained in current RM cell, and $Time_{old}$ is the time interval field contained in the previous RM cell. (The constant k_0 will be explained later.) The product is the number of cells that were requested in the previous RM time interval Req_{old}, which is then added to the register Q_{reg}. In this way, cells that have been requested in the previous feedback interval can be added to the effective queue size estimate.

At the same time, the QRC in each switch maintains a register Ex_{reg} for all the

unsatisfied cell requests from downstream switches (other switches in the direction of the destination). These "excess" cells are the cells that have been requested from downstream but cannot be sent because of a bottleneck at the node. The requested cells at the node Req_{old} are subtracted from the cells requested at the previous node Req_{rm}. This difference (which represents the number of QRC cells that cannot be requested at this stage) is added to Ex_{reg} . Then Req_{old} is written into Req_{rm}. The calculated QRC_{new} is then written into QRC_{rm} before the RM cell leaves the switch. We will explain later how the algorithm computes the new QRC for the switch.

When a *forward* RM cell leaves a switch, Ex_{reg} is added to Req_{rm}. Ex_{reg} is then reset to zero. We can think of Req_{rm} as the number of cells that have been requested previously by the switch downstream. When a forward RM cell reaches a switch, Q_{reg} is decremented by Req_{rm}. This operation maintains a count of how many cells requested from the switch are still in the network.

The following pseudocode summarizes the operations performed by the QRC algorithm to estimate the effective queue size:

Backward RM cell arrives at the switch:

```
QRC_BACK_RM()
    {
    Req_old = min(Time_rm, k0 Time_old) × QRC_old
    Q_reg = Q_reg + Req_old
    Ex_reg = Req_rm − Req_old
    Req_rm = Req_old
    QRC_rm =CALCULATE_NEW_QRC()
    QRC_old = QRC_rm
    Time_old = Time_rm
    }
```

Forward RM cell arrives at the switch:

```
QRC_FORW_RM()
    {
    Q_reg = Q_reg − Req_rm
```

$\mathsf{Req_{rm}} = \mathsf{Req_{rm}} + \mathsf{Ex_{reg}}$

$\mathsf{Ex_{reg}} = 0$

}

Note that if the propagation delays for the backward RM cells vary or if some RM cells are lost, some error will be introduced in our estimate of the effective queue size. In the first case (time-varying propagation delay), the error can be corrected by using an extra field in the next RM cell. This extra field records the error in the QRC request of the previous RM cell and then corrects the effective queue size at the affected nodes. Likewise, in the second case (RM cell loss), the accumulated error can be corrected by periodic RM cells that compare the total number of QRC requests that each node has made and received. The discrepancies can be used to correctly update the effective queue sizes. The implementation details of this error recovery scheme are beyond the scope of this thesis.

Computing the New QRC Rate

In order to calculate the new QRC rate, we use a proportional controller which takes the effective queue size as its input. The effective queue size is the sum of the residual and real queue sizes ($Q_{reg} + Q_{real}$). We use the effective queue rather than the real queue to perform our calculations because there is no delay between the time a cell is requested and when it enters the effective queue. This eliminates the instability associated with delays in feedback control loops. The gain of the controller is determined by the time interval supplied by the backward RM cell. In pseudocode the function to calculate QRC is:

```
CALCULATE_NEW_QRC()
```

}

$\mathsf{QRC_{temp}} = \frac{\mathsf{THRESH} - \mathsf{Q_{reg}} - \mathsf{Q_{real}}}{\mathsf{k_0\,Time_{rm}}}$

Return $\max[\min((\mathsf{QRC_{rm}} + \mathsf{RRC_{rm}} - \mathsf{Available\ Rate}), \mathsf{QRC_{temp}}), -\mathsf{Available\ Rate}]$

}

THRESH is the desired queue size. It should be set to zero or close to zero. Any queue size which will not cause significant queuing delay is adequate. In steady state, the queue of the slowest switch in the VC path will be of this size. The choice for THRESH is not a real design consideration since any small value (relative to the total queue capacity) will do.

The parameter k_0 is a constant gain factor. Its value should be greater than 1 and less than approximately 10. The proportional controller will calculate the rate so that $1/k_0$ of the queue error will disappear in one feedback step. Trying to eliminate the entire queue error in one step can create oscillations due to measurement errors. Setting the goal for less than one tenth of the error unnecessarily slows down the system. Simulations show that a value of about 5 gives smooth and fast results. Again, the exact choice of k_0 is not critical, and any value within a wide range is good.

The rate command calculated by the controller (QRC_{temp}) is limited so that the total rate command (QRC+RRC commands) is not larger than the total rate command downstream *and* so that the total rate command is not negative.

5.1.5 Source Behavior

When the source receives a backward RM cell at time t_{arrive}, it sets its data sending rate to be the sum of the QRC and RRC commands ($QRC_{rm} + RRC_{rm}$). If the source cannot or does not want to comply with this rate request, it can send data at any rate below the specified rate.

The backward RM cell is then forwarded to the data destination. The data fields in the RM cell are left untouched. Even if the source sends less data than the requested rate, the effective queue size estimates will be correctly updated, and the overall algorithm still works.

The source will also set an expiration interrupt for time $t_{arrive} + k_0 \times Time_{rm}$. If the next RM cell arrives before the expiration time, the rate values of the new RM

cell will become the current data sending rates. On the other hand, if a new RM cell does not arrive by the expiration time, the source stops sending data. When a new RM cell arrives, the source restarts transmitting data at the rate specified by the RM cell.

5.2 Analysis

In this section, we will analyze the behavior of the control algorithm. We first look at the single node case and see how the minimum time requirements are satisfied. Then we will show its stability and fairness in the multi-node case. In this section, we still assume that all the switches use a per-VC queuing discipline.

5.2.1 Single Node Case

In figure 5-1, we present a block diagram representation of the feedback loop for the single node case. We use a discrete-time dynamic system model. The time interval between events is the inter-arrival time between RM cells at the switch (Δ_{rm}). At time n, $r[n]$ is the number of cells serviced, $d[n]$ the error in measuring the available rate, $x[n] =$ `THRESH` the desire queue level, and $q[n]$ the real queue size. For convenience, we shall express the RTD in terms of integral multiples of Δ_{rm}, which is assumed to be small compared with the RTD.

This model is similar to the continuous time model presented in section 3.1. The main difference is that the secondary delay (the lower of the two in the figure 5-1) used in the QRC is not an artificial delay in a Smith Predictor but the real delay of the network. If the real round-trip delay varies, the secondary delay in the QRC will vary simultaneously. Conceptually, the primary delay models the delay in carrying the real data, whereas the secondary delay models the delay in carrying the RM cells which describe the QRC traffic. We showed in section 3.2 how the QRC algorithm can be thought of as an observer of the RRC error. The command $u_q[n]$ is an observer estimate of the difference between the output and input rates at the queue, *if* the RRC were to function by itself. With the new algorithm, the QRC uses the information

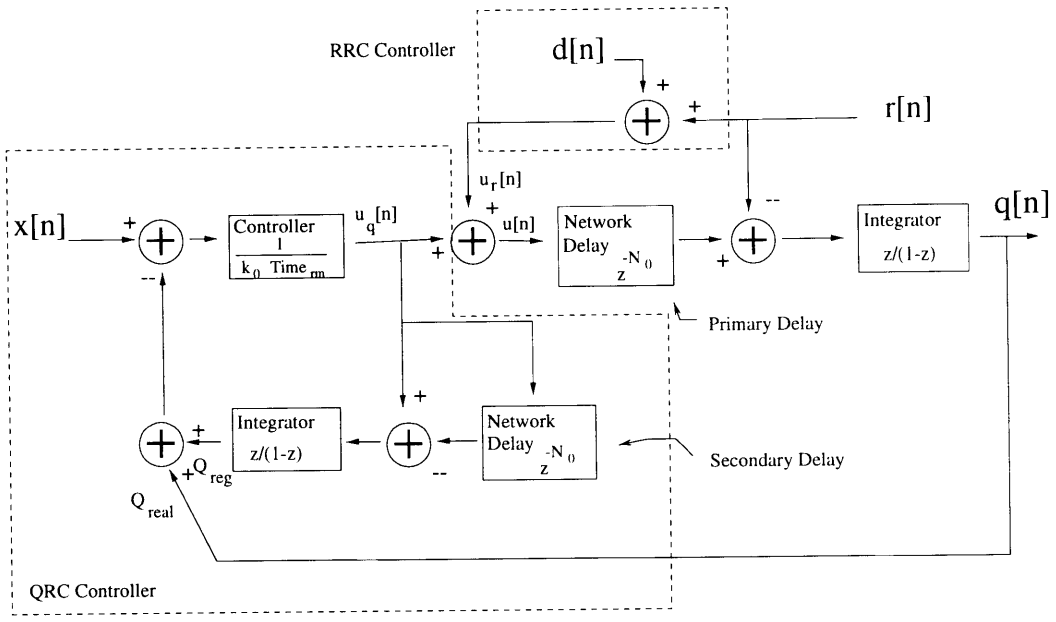

Figure 5-1: Single Node Block Diagram

contained in RM cells to *simulate* the flow of the real data cells it has requested, and thus being able to observe and correct the error of the basic RRC algorithm.

Using z-transform methods, we can calculate the transfer functions from x,d,and r to q.

$$H_x(z) = \frac{Q(z)}{X(z)} = z^{-N_0} \frac{K}{(K+1) - z^{-1}} \tag{5.1}$$

$$H_r(z) = \frac{Q(z)}{R(z)} = \left(-1 + z^{-N_0}\right) \frac{1}{1 - z^{-1}} + \left(z^{-N_0} - z^{-2N_0}\right) \frac{K}{(1 + K - z^{-1})(1 - z^{-1})} \tag{5.2}$$

$$H_d(z) = \frac{Q(z)}{D(z)} = z^{-N_0} \frac{1}{1 - z^{-1}} - z^{-2N_0} \frac{K}{(1 + K - z^{-1})(1 - z^{-1})} . \tag{5.3}$$

N_0 is the number of discrete steps in the round-trip delay, and K is the gain of the proportional controller, $1/(k_0 Time_{rm})$.

From the transfer functions, we can see that every input-output relation is stable. A bounded error in the switch rate measurement will only cause a bounded error in the real queue size. A change in the service rate will only affect the queue size for

some transient time.

From equation (5.2), we can calculate how soon $q[n]$ will reach steady state after $r[n]$ reaches steady state. This is what we referred to in section 2.5 as the settling time or time to stabilize the queue. By using inverse z-transforms, we can find the impulse response $h_r[n]$ between $r[n]$ and $q[n]$:

$$\begin{aligned} h_r[n] = -u[n] + (2 - (1+K)^{-(n+1-N_0)})u[n-N_0] \\ -(1 - (1+K)^{-(n+1-2N_0)})u[n-2N_0] \ , \end{aligned} \tag{5.4}$$

where $u[n]$ is the step function. Since $K = 1/(k_0\ Time_{rm})$, as the interval between RM cells decreases (the feedback frequency increases), the gain K increases and $h_r[n]$ approaches zero for $n > 2N_0$ or $2RTD$. Furthermore, $\sum_{n=0}^{\infty} h_r[n] = 0$. This leads us to the following theorem:

Theorem 4 Suppose a switch is connected to a single source and the bi-channel rate control (BRC) algorithm is used. At two round-trip delay time (RTD) following a change in the available rate, the queue length of the switch can be made arbitrarily close to the reference level by increasing the feedback frequency. Moreover, given a fixed feedback frequency, the error between the queue length and the reference level will converge to zero exponentially fast after 2 RTD following a change in the available rate.

Proof: By simple substitution into equation (5.4), it is easy to see that with a sufficiently high feedback frequency (so that K is large), the value of $h_r(2RTD)$ can be made arbitrarily close to zero. The values of h_r after $2RTD$ will exponentially decrease with n. Therefore, $\sum_{n=0}^{2RTD} h_r[n] \approx \sum_{n=0}^{\infty} h_r[n] = 0$. In other words, $\sum_{n=0}^{2RTD} h_r[n]$, which is the response of $q[n]$ to a step change in $r[n]$, can be made negligible at 2 RTD by increasing the feedback frequency. By linearity, the total response $q[n]$ is the sum of the responses to $x[n]$ and $r[n]$ (assuming we have a correct measurement of the available rate, i.e. $d[n] = 0$). But the response to $x[n]$ (a constant) is

THRESH. Hence, $q[n]$ can be made arbitrarily close to its steady-state value (THRESH) at 2 RTD.

After 2 RTD, as $\sum_{i=0}^{n} h_r[i]$ approaches zero exponentially, $q[n]$ will approach THRESH exponentially. □

Note that the value of $h_r(2RTD)$ decreases *exponentially* with the feedback frequency. Therefore, the difference between $q[n]$ and THRESH at 2 RTD will also decrement exponentially with frequency. A small increase in feedback frequency will result in a large decrease in the queue size error at 2 RTD. A feedback period that is an order of magnitude smaller than the RTD will be enough to ensure that the queue size at 2 RTD will be within a few percentages of the reference level. In theorem 2.5, we established that the minimum time to stabilize a queue without knowledge of the round-trip delay is 2 RTD. Therefore, the BRC algorithm approaches this limit exponentially as the feedback frequency increases.

5.2.2 Multiple Node Case

Transient Response

The behavior of our control algorithm is not very different in the multiple node case from the single node case. The slowest or most congested node of a VC path controls the dynamics of the VC flow. This node (the *active* node) imposes its rate commands over the other nodes. The dynamics of the VC queue at the active node are similar to those for the single node case. As in the single node case, the multi-node algorithm will stabilize effectively the queue level of its active node. In steady state, the active node will have a queue size equal to THRESH. The other nodes in that VC will have empty queues.

After the available service rate of a particular switch in a multi-node network achieve steady state, it will take 2 RTD (as defined in section 2.5) to stabilize the queue of the switch. We are assuming that the feedback frequency is sufficiently high. During the first RTD, the new RRC command is being sent to the source and the switch is expecting this new rate. At 1 RTD, the switch starts receiving data at the

correct RRC rate. Since the effective queue is now stable, the switch stops requesting a QRC command and the total command sent by the switch achieves steady state. After another RTD, the arrival rate at the switch reaches steady state and the whole system is in steady state. As pointed out in section 2.5, this is the minimum time for a flow control algorithm without knowledge of link delays and without communication between VCs.

Stability

In the last section we saw that the transmission rate of the source is limited by the sum of the RRC and QRC commands. The RRC command is simply the minimum service rate of all the switches in the VC path. Therefore, by itself, the RRC at most can only prevent the queue of the slowest node from decreasing.

On top of that, the QRC during one feedback interval Δ_{rm} can at most request enough cells to *partially* fill the effective queue size to `THRESH`. The expiration at the source makes sure that this number of cells is never exceeded. Since there is no delay between the time when a QRC cell is requested and when it enters the effective queue, there is no possibility for overshoot. Each switch knows how many of its QRC requests are still upstream (residual queue) and will not request an excess of cells so as to create cyclic overshoot and instabilities. Because the queue sizes in a particular VC cannot overshoot, our algorithm is stable. A bounded change in the available rates will only cause a bounded change in the actual rate and a temporary change in the queue sizes. Since the algorithm does not use a Smith predictor to compute the residual queue, its estimate of this queue is correct even when the round-trip delay varies. Unlike the primal algorithm in section 3.1, our new algorithm is stable even under changes in the round-trip delay.

Even though the dynamics of the independent VCs are stable, we should make sure that the interaction between the different VCs does not give rise to instabilities. Seen from a particular node i, a decrease in the rate of any one of the VCs passing through it may cause an increase in the available rate of the other VCs passing through node i, which will cause the VCs active at node i (VCs whose slowest node is node i) to

increase their actual rates. However, because of per-VC queuing, an increase in the actual rate of a VC passing through node i will decrease the available rate of only the VCs *active* at node i, which will be forced to reduce their actual rates. The forced reduction of the rate of a VC active at node i will *not* cause any change in the available rates of other VCs passing through node i. Therefore, after a few round-trip delays the VCs passing through node i will stop interacting and they will be stabilized. Other nodes in the network might also be affected by changes in available rate, but they will also stabilize a few round-trip delays after they are disturbed. If there are no loops in the network, the entire network is guaranteed to eventually stabilize. In the unlikely event that loops do exist, global stability cannot be proved, and the outcome depends on the topology of the network. Nevertheless, for most reasonable networks, the rate variations die out as they propagate to different VCs.

Fairness

In steady state, the source follows the command of the slowest or active node of the VC path. Therefore this node will have a queue size equal to `THRESH`. The other nodes in that VC will have empty queues. We can then make the following statement on the algorithm's allocation of rates to the different VCs.

Theorem 5 The bi-channel rate control (BRC) algorithm provides a max-min fair allocation of bandwidth among the different virtual channels.

Proof: In steady state, each virtual channel will include at least one node (the active node) where the queue size is equal to a reference level. Since the service discipline is round-robin, the active node of a particular VC will service that VC no slower than all the other VC's passing through that node. Therefore, the link following the active node of some VC is a bottleneck link for that VC. Since every VC has a bottleneck link, the overall rate allocation is max-min fair. □

5.3 Virtual Queuing

In the preceding sections, we have discussed the implementation of the BRC flow control algorithm in network switches that use per-VC queuing. The advantage of this queuing discipline is that it decouples the dynamics of one VC from the other VCs. In per-VC queuing, each VC sees the combined effects of the dynamics of the other VCs as simple variations in its available rate.

In this section, we show how to implement the BRC flow control algorithm in network switches that use FIFO queuing. We accomplish this by using the technique of *virtual queuing* [6, 26, 27], which emulates per-VC queuing on a FIFO queue. Simply put, the idea involves keeping track of the queue length on a per-VC basis as if per-VC queuing were actually being used. The operation of per-VC queuing can be summarized in the following pseudo-code. $q^i_{virtual}$ is the virtual queue size for VC i, `MAXVC` is the total number of VCs passing through the switch, and i is a local variable.

if a cell arrives from VC j,

$q^j_{virtual} = q^j_{virtual} + 1$

if a cell is serviced by the FIFO queue,

```
do
    {
    i = i + 1
    if (i = MAXVC)
        i = 0
    }while(q^i_virtual = 0)
q^i_virtual = q^i_virtual - 1
```

While the real data cells are stored in the FIFO queue, the RRC and QRC algorithms pretend that the cells are stored in the (virtual) per-VC queues. They perform all their calculations as if the switch was still using per-VC queuing. Therefore, the rates the QRC and RRC algorithms request will still be the same as in the per-VC

queuing case. After the round-trip delay, the inputs to the FIFO queue will become these rates. After the queuing delay, the output of the FIFO queue will be on average (with averaging window of size greater than the round-robin period) the same as that of a per-VC queue. Therefore, as long as the rate requests are based on a per-VC queue (real or virtual) after a sufficient time period (one round-trip delay and one queuing delay), the behavior of the FIFO queue will be on average (as defined above) the same as that of a per-VC queue.

Note that the virtual queue is not used as an estimation of the real queue, but as a tool to distribute bandwidth fairly among the various VCs. The key idea is to request data from the source as if the switches used per-VC queuing. Once the correct rates are sent from the sources, the VC rates across the network will converge to the equivalent rates under per-VC queuing. The simulation results presented in the next section will show that the virtual queuing technique does behave very similarly to real per-VC queuing.

5.4 Simulations

In this section we present some simulation results using our flow control algorithm in a network with per-VC queuing and in networks with FIFO queuing. Both scenarios were simulated using the Opnet simulation package.

5.4.1 Network Topology and Simulation Scenario

The network configuration that we are simulating is the same as the one used in subsection 4.3.2 and is depicted in figure 5-2. Once again, sources 0, 1, 2, and 3 are sending data to receivers 3, 0, 1, and 2 respectively. The propagation delays between the switches is 10 msec, and that between the switches and users is 0.5 usec. Switches 0, 1, 2,and 3 can transmit data at each of their ports at rates 7000, 8000, 9000, and 10,000 cells/sec respectively. The RM cell frequency is set to 1/64 of the maximum data rate or 1562.5 Hz, and the QRC gain is set to $1/(5\Delta_{rm})$ or 312.5 s^{-1}. The desired queue level `THRESH` is 50 cells for each of the VC queues.

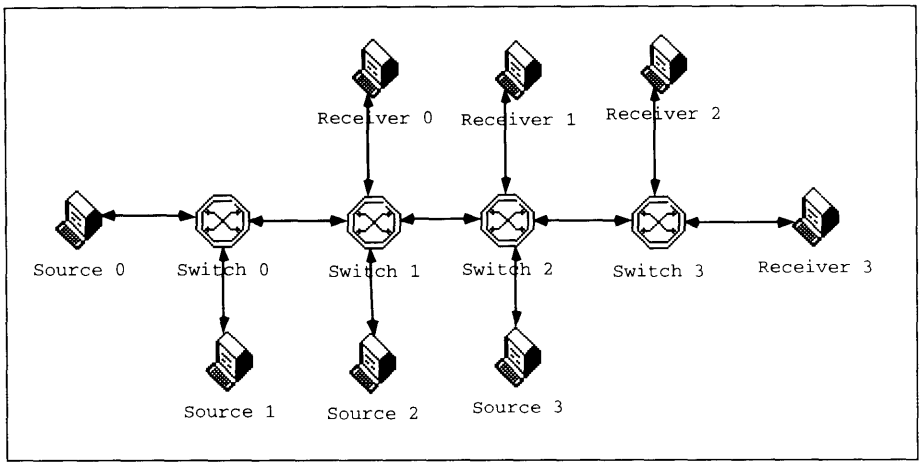

Figure 5-2: Multiple Node Opnet Simulation Configuration

At initialization, none of the sources are sending any data. The receivers are continuously sending RM cells to the sources at the fixed frequency. At time 0.05 sec, source 0 starts sending data to receiver 3 on VC 3. Because switch 0 is the slowest node, it becomes the bottlenecked and active node. At time 0.15 sec, source 3 starts sending data to receiver 2 on VC 2. Because switch 2 has to transmit cells belonging to VCs 2 and 3, it becomes the next active node. At time 0.3 sec, source 2 starts sending data to receiver 1 on VC 1. This causes switch 1 to become the next active node. Finally, at time 0.4 sec, source 1 starts sending data to receiver 0 on VC 0. Switch 0 becomes once again the active node.

5.4.2 Network with per-VC Queuing

When the network has a per-VC queuing discipline, our algorithm uses the per-VC queues to perform its RRC and QRC calculations. Our new algorithm, without using knowledge of the round-trip delay, can achieve practically identical results to the algorithm simulated in chapter 4, which did use knowledge of the round-trip delay.

In figure 5-3, we can observe the RRC commands that the switches compute for the sources. The data samples are recorded at the time when the RM cell with the command arrives at the corresponding source. Initially, the RRC command for source 0 is 70,000 cell/sec, which is the rate of the slowest link in the path. When

source 3 starts sending data, the RRC command for both sources becomes 45,000 cells/sec, which is half the rate of the node they share (switch 2). When source 2 starts transmission, the RRC commands for sources 0 and 2 become 40,000 cells/sec, since that is half the capacity of the node they share (switch 1). Since source 0 is only using 4/9 of switch 2's capacity, the RRC command for source 2 is increased to 5/9 of switch 2's capacity or 50,000 cells/sec. Finally, when source 1 starts sending data, the RRC command for sources 0 and 1 becomes 35,000 cell/sec (half of switch 0's capacity), and sources 2 and 3 increase their rates to 45,000 and 55,000 cells/sec respectively to utilize the unused bandwidth in switches 1 and 2.

The RRC commands shown in figure 5-3 are max-min fair to all the sources. However, we can also see that it takes some time (delay between switch and source) for a new RRC command computed at the switch to be followed by the source. For example, at time 0.15 sec, switch 2 send a lower RRC command to source 0. However, the command reaches source 0 at time 0.17 sec. Between times 0.15 and 0.19 sec, switch 2 is receiving from sources 0 and 2 more cells that it can service. This error cannot be corrected by the RRC controller since latency is an inherent characteristic of the network. However, this error can be corrected at a later time by the QRC controller.

In figure 5-4, we can observe the QRC commands that arrive at each source. Initially, when each source starts sending data, the initial QRC command is high in order for the source to send an initial burst of cells that will fill up the VC queue of the active node to the desired size (50 cells in this case). After the initial burst, the QRC source quickly settles down to close to zero.

At time 0.17 sec the QRC command for source 0 goes negative in order to slow source 1 down. This is needed because source 3 has started sending data at time 0.15 sec and switch 2 has become congested. Since the QRC controller in switch 2 can observe the error caused by the delay, it can request the exact QRC rate needed to correct this error. Likewise, at time 0.31 sec, the QRC command for Source 0 becomes negative to reduce the congestion at switch 1 caused by the activation of source 1. Throughout the simulation, the small QRC commands correct the small

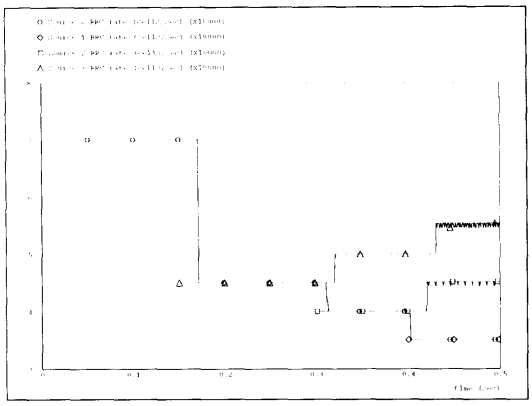

Figure 5-3: RRC Commands at Each Source with per-VC Queuing

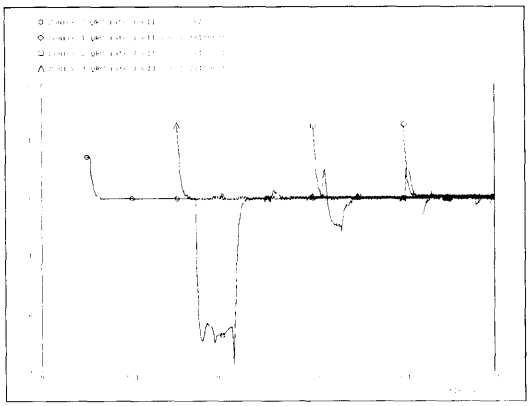

Figure 5-4: QRC Commands at Each Source with per-VC Queuing

errors in the RRC due to the errors in the estimation of the actual service rate.

In figure 5-5, we can observe the queues for VC 3 (the VC between source 0 and receiver 3) at each of the switches. At time 0.05 sec, the queue of the active node (switch 0) fills up to the desired queue size of 50 cells. Notice how the ramp is close to a first order exponential without any oscillations. At time 0.15 sec, when source 3 starts sending data, switch 2 becomes the active node and its queue starts increasing steadily for 0.04 sec (round-trip delay). Notice how it takes **2 RTD** (0.08 sec) to stabilize this queue back to the 50 cell level.

At time 0.3 sec, switch 1 becomes the active node, and its queue increases for 0.2 sec (round-trip delay). The QRC command forces the queue size to decrease to the desired level. Again, it takes **2 RTD** (0.04 sec) to stabilize this queue. At time 0.4 sec, switch 0 becomes the active node again. Since the delay to source 0 is negligible, there is no overshoot. Note that when a node ceases to be active, its queue empties out. At every moment of time between 0.05 and 0.5 sec there is at least one active node with a non-empty queue.

In figure 5-6, we can observe the VC queues for VCs 0,1, and 2. For each VC there is one queue in its path that is of size 50 (the desired queue size). Each queue fills up as soon as the corresponding VC starts transmitting data. For these VCs the queue sizes remain mostly stable throughout the simulation because of the negligible round-trip delay.

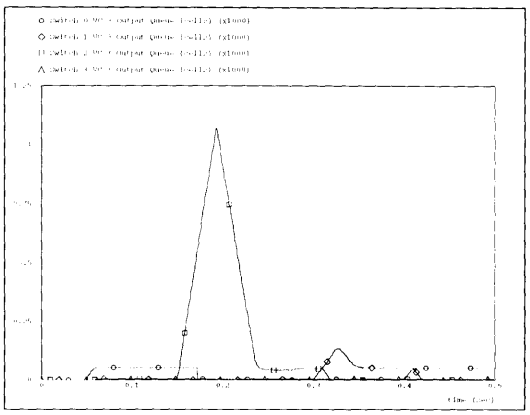

Figure 5-5: Output Queue of VC 3 at Each Switch

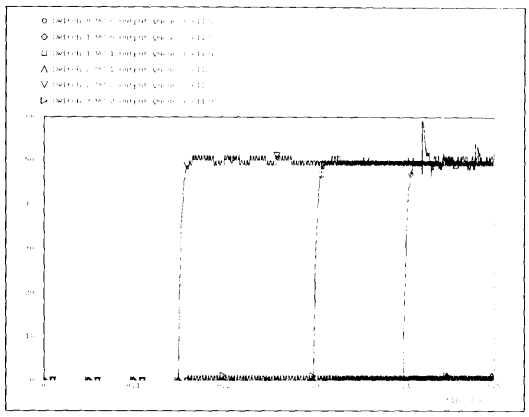

Figure 5-6: Output Queues of VCs 1, 2, and 3 at Each Switch

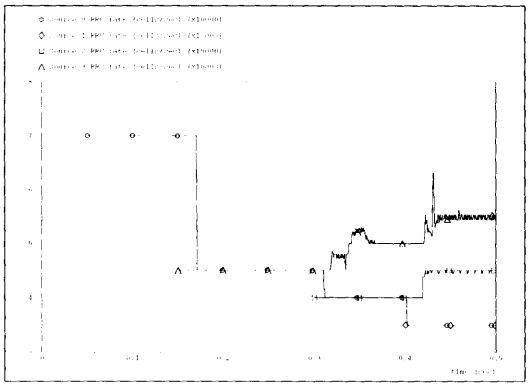

Figure 5-7: RRC Commands at Each Source with FIFO Queuing

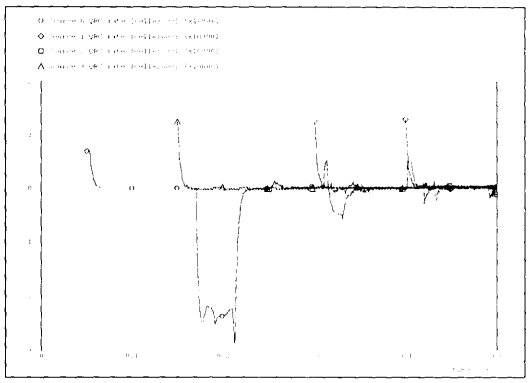

Figure 5-8: QRC Commands at Each Source with FIFO Queuing

5.4.3 Network with FIFO Queuing

When the network uses a FIFO queuing discipline, our new algorithm needs to make use of virtual queuing to obtain information for the RRC and QRC controllers. Surprisingly, the performance of the algorithm with virtual queuing is almost identical to its performance with per-VC queuing.

In figure 5-7, we notice a slight degradation of the RRC commands. Nevertheless, the RRC command is still max-min fair on average and during steady state. In figures 5-8, 5-9, and 5-10 we notice that the QRC commands and the virtual queue levels are practically identically to the QRC commands and per-VC queue sizes of the previous scenario. Finally, in figure 5-11, we can observe the FIFO queue sizes of each switch and notice how they are stable and controllable.

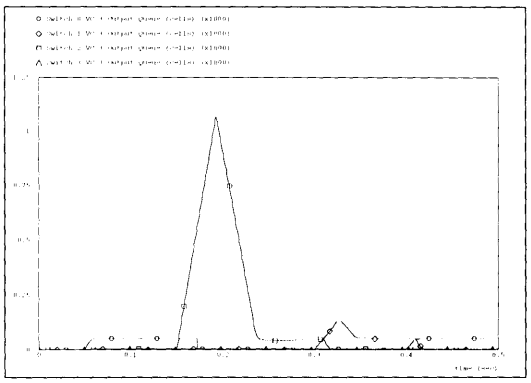

Figure 5-9: Virtual Queues of VC 3 at Each Switch

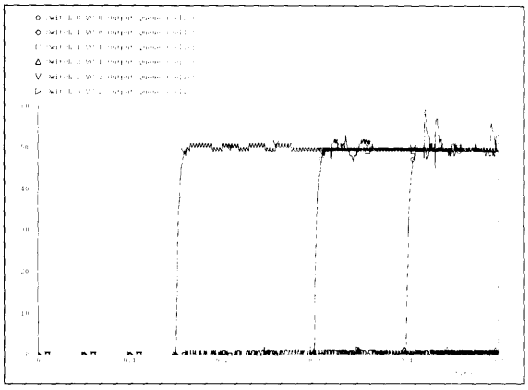

Figure 5-10: Virtual Queues of VCs 1, 2, and 3 at Each Switch

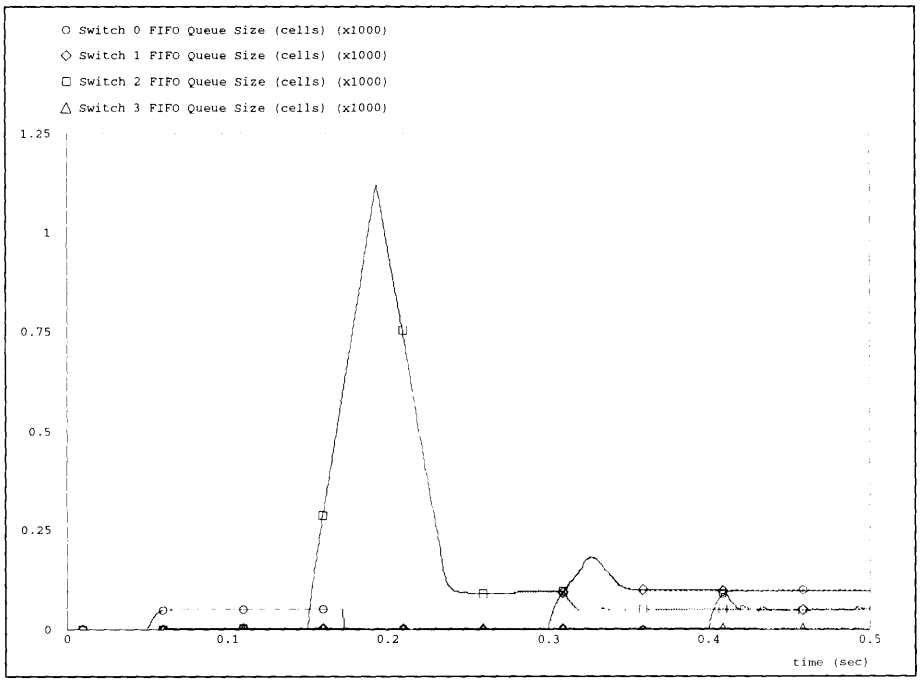

Figure 5-11: Actual FIFO Queues at Each Switch

Chapter 6

TCP Flow Control

6.1 Objective

While ABR and the algorithms we have discussed so far assume that the end systems comply with the explicit rate command, most current applications are only connected to ATM via legacy networks such as Ethernet. In fact, simulation results reported in recent ATM Forum contributions [10, 28] suggest that flow control schemes in ABR relieve congestion within the ATM network at the expense of congestion at the network interfaces (network edges) and cannot provide adequate flow control on an end-to-end basis between data sources. At the same time, most data applications today employ the Transmission Control Protocol (TCP), which provides flow control at the transport layer. The need for ABR to provide flow control at the ATM layer is questionable in light of TCP's built-in flow control mechanism which works independently of the ATM layer control.

This chapter presents a new efficient scheme for regulating TCP traffic over ATM networks with the goal of minimizing the network-interface. The key idea underlying this scheme is to match the TCP source rate to the ABR explicit rate by controlling the flow of TCP acknowledgments at network interfaces. We present analytical and simulation results to show that this scheme has minimal buffer requirement, yet offers the same throughput performance as if the network interface buffer were infinite. Moreover, the scheme is transparent to the TCP layer and requires no modification

in the ATM network except at the network interface.

6.1.1 TCP over ATM

Transmission Control Protocol (TCP) is a connection-oriented transport protocol which is designed to work with any underlying network technology. Because it makes no assumption on how the network processes the data it sends, TCP must perform its own data recovery and flow control algorithms.

The flow control mechanism is meant to slow down TCP when the network becomes congested. TCP has no direct way of knowing when the network is congested. It can only indirectly detect congestion by keeping track of how many packets are lost. When packets do get lost, the loss indicates that some queue in the network might have overflown. Every time TCP detects a packet loss, it reduces its rate to alleviate the congestion that could have caused the packet loss.

TCP's congestion control (as well as error recovery) are implemented by a dynamic window at the source. Every packet that is sent must be acknowledged by the source. The window size dictates the number of unacknowledged packets that can be present in the network at any given time. When a new packet is acknowledged, the window size increases, and when a packet is lost, the window size decreases, forcing the rate to decrease [9].

In a high-latency network environment, the window flow control mechanism of TCP may not be very effective because of the time it takes for the source to detect congestion. By the time the source starts decreasing its rate, the network has been congested for some significant amount of time. Ideally, we would want the source to react to congestion before it occurs rather than acting when it is too late.

When integrating TCP with ATM-ABR, we run into the problem that TCP at each source ignores ATM-ABR's explicit rate. Even though the network layer knows how much rate the source can send at a given time, TCP has its own dynamics and will be sending data at different rates depending on packet loss and its end-to-end communication.

The classical solution to the problem is to place a buffer in the network interface

or edge device between TCP and ATM. If TCP is sending data at a faster rate than the ATM's explicit rate, this buffer will hold the extra packets that TCP is sending.

This brute force approach, although being relatively simple, hardly takes advantage of the effective congestion control schemes developed for ABR service. Even though the ATM network might never be congested, the buffer between TCP and ATM can be in a constant state of congestion. As suggested by the simulation results in [10, 28], improving ATM's congestion control will only displace more congestion from the network to the edge device.

6.1.2 Acknowledgment Bucket

In this paper, we propose a radically different solution. Instead of holding packets at the edge buffer, we will slow down the TCP source by holding the acknowledgments that it will receive. In this way, we are transforming a packet buffer at the output of the edge device into an acknowledgment "buffer" at its input. Since the essential information contained in an acknowledgment is only a sequence number, this acknowledgment "buffer" is really only a list of numbers. To differentiate this type of buffering from the traditional packet buffer, we will call it *acknowledgment bucket.*

We will show that by adequately regulating the acknowledgment bucket, we can practically eliminate the packet buffer and still achieve the same output rate at the edge device. The advantage is that while storing one packet in the buffer might require a few Kbytes of memory, storing its acknowledgment in the bucket requires only a few bytes (size of the sequence number) of memory.

By controlling the flow of acknowledgments back to the source, *we are effectively extending the explicit rate flow control of ABR all the way to the TCP source.* The acknowledgment bucket serves as a *translator* which transforms the ABR explicit rate command into a sequence of TCP acknowledgments whose effect is to have the TCP source send data no faster than the ABR explicit rate.

The acknowledgment bucket is analogous to the storage of permits in the leaky bucket scheme. TCP acknowledgments serve as permits that allow the edge device to request packets from the source. When the acknowledgment bucket is empty, the

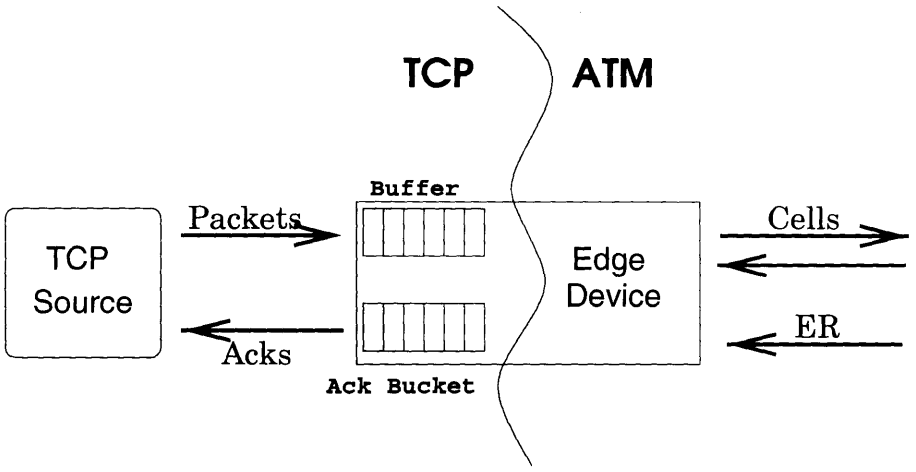

Figure 6-1: General Model

edge cannot request any more packets. The acknowledgment bucket gets filled up according to TCP dynamics and is drained out according to the ABR explicit rate.

In section 6.2 we will lay out the specific algorithm used at the edge device. In section 6.3, we analytically solve for the dynamics of our new scheme and compare them to the conventional scheme. Finally, in section 6.4, we simulate our new scheme as well as the conventional scheme on a simple network to illustrate our theoretical claims.

6.2 Algorithm

6.2.1 Model

The fundamental principle behind our acknowledgment bucket scheme is based on the ability of the edge device to withhold acknowledgments returning to the source. The edge device will only release enough acknowledgments so that the source will send data no faster than the ATM explicit rate.

We assume that the acknowledgment travels on its own. If the acknowledgment is contained in a data packet traveling in the opposite direction, we can always separate the two into two different packets so that the acknowledgment can be withheld without stopping the data.

As seen in figure 6-1, the edge device sends and receives cells from the ATM network. It also receives packets and sends acknowledgments to the TCP source. We assume that each TCP connection has only one edge device through which it can send data. The packets that have been received from the source and have not yet been sent to the ATM network are stored in the packet buffer. The acknowledgments that have been received from the destination through the ATM network and have not yet been sent to the source are stored in the acknowledgment bucket. The edge device receives an explicit rate (ER) command from the ATM network. This ER is used to determine the rate at which acknowledgments can be released from the bucket.

In order to determine the forwarding rate for the acknowledgments, the edge device needs to know the effect of a released acknowledgment on the source. If the TCP source has data to send in the time interval of interest, it will react to the arrival of an acknowledgment by transmitting one or two packets. This *ack response* depends exclusively on the congestion window size when the acknowledgment reaches the source. In order to predict the response of an acknowledgment, the edge device must keep track and predict the size of the congestion window at the source when the acknowledgment will reach the source. This window size prediction is done with an observer at the edge device.

6.2.2 Observer

An observer is an artificial simulator of the dynamics of a system and is used to estimate the states of the original system. In our case, we are interested in knowing the congestion window size at the TCP source. We obtain this state by simulating the window dynamics in the edge device itself.

The dynamics of the congestion window at the TCP source are relatively straightforward. First, when the window size w is less than W_{mid}, the window size increases by one every time a new acknowledgment arrives at the source. This is called the *slow start* phase. When w is between W_{mid} and some maximum W_{max}, the window size increases by one every time w acknowledgments arrive at the source. This is called the *congestion avoidance* phase. Finally, when w reaches the maximum rate W_{max},

w stops changing as long as packets do not get lost. We will call this the saturation phase.

The observer works by predicting these window dynamics. When an acknowledgment is released by the edge device, the observer will predict what the window size at the source will be when *that* acknowledgment is received by the source. The observer can be implemented by the following subroutine at the edge device which is invoked every time an acknowledgment is released.

Acknowledgment leaves the bucket:

```
WINDOW_OBSERVER()
    {
    if (w < W_mid)
        w = w + 1
        return(1)
    elseif (W_mid ≤ w < W_max)
        w_frac = w_frac + 1
        if (w_frac == w)
            w = w + 1
            w_frac = 0
             return(1)
        else
            w = w
            return(0)
    else
        w = w
        return(0)
    }
```

Apart from keeping track of the window size w, the subroutine also returns the increase in window size caused by the release of the last acknowledgment. During

initialization, w is set to one, W_{max} is set to the maximum window advertised by the destination, and W_{mid} is set to half of that.

6.2.3 Translating Explicit Rate to Ack Sequence

Once the edge device knows how the window size is increasing at the source, it can start translating the ATM explicit rate into a sequence of acknowledgment releases. The edge device receives periodically a resource management (RM) cell from the ATM network that contains the explicit rate command.

We implement this rate translation by keeping track of the number of packets R that have been successfully requested by the ATM network as well as the number of packets U that have been requested from the source through acknowledgment release. The ATM request R is increased at the rate specified by the ER as long as the acknowledgment bucket is nonempty. If there is no acknowledgment left, the explicit rate cannot be satisfied by the source and therefore should not be accounted. The TCP request U is increased by one every time an acknowledgment is released. Furthermore, if the observer function returns a one (the window size will increase at the source), U is incremented by another packet. Whenever $R > U$, the edge device will release an acknowledgment from the bucket to keep R and U roughly equal.

The following pseudocode demonstrates how the translator can be implemented:

```
while # acks > 0
    {
    R = R + ER × (time() − τ)
    τ = time()

    if (R > U)
        RELEASE_ACK()
        U = U + 1
        if ( WINDOW_OBSERVER() == 1)
            U = U + 1
```

}

The number of acknowledgments available in the bucket is denoted by # acks, and the present time is denoted by time(). Note that data has been quantified for convenience in terms of number of packets. If the packets have different sizes, the same ideas still hold by using the TCP octet rather than the IP packet as the data unit.

6.2.4 Error Recovery

Our discussion of the acknowledgment bucket scheme so far assumed that there was no packet loss in the network. However, when there is packet loss in the network, the source congestion window behaves in a more complicated way than described in subsection 6.2.2, and our original observer will need modification.

The TCP source holds a timer for every packet that it sends in order to estimate the roundtrip delay between itself and the destination. Based on the roundtrip delay and its variance, the source computes the maximum time interval (RTO) within which a previously sent packets can remain unacknowledged. If a packet remains unacknowledged for more than RTO, the source expires, reducing the window size to one packet and resending the first unacknowledged packet. If the TCP source has fast retransmit and it receives duplicate acknowledgments (typically four of the same kind), it will also decrease its window size (to one packet in TCP Tahoe) and resend the first unacknowledged packet.

Instead of trying to simulate all these complicated dynamics in our observer, we can use a simple measurement to signal our observer when it should change phase. This is done by a detector at the edge device which reads the sequence number of the packets arriving from the source. During normal operation the sequence number should be in consecutive order. If the sequence number of some packet received at the edge device is the same as that of some packet previously received, it indicates that the source is retransmitting. In this case the detector triggers the observer to reset its states. The following pseudocode shows how the observer should be reset

after the repeated packet detector gets triggered:

Repeated packet is detected:

```
RESET_OBSERVER()
    {
    W_mid = w/2
    w = 1
    }
```

Note that this reset function is valid only if the source is using TCP Tahoe. If the source uses a version which uses fast recovery (TCP Reno), the reset function would have to be modified accordingly.

6.2.5 Extension

This acknowledgment holding scheme can be used by any network (not necessarily ATM) trying to regulate the flow from a source that responds to information from the destination. By finding out how the source transmission rate reacts to the information sent by the destination, the network can manipulate such information (including withholding it) to effectively force the source to send data at the network's desired rate.

6.3 Analysis

In this section we will analyze and compare the dynamic behavior of TCP with and without an acknowledgment bucket. To simplify our analysis, we will use a fluid approximation. For cases where the ATM link is highly utilized, the fluid approximation is quite accurate since the system does not exhibit the burstiness characteristic of TCP when the link is underutilized. It is these cases that are of most interest to us because it is only in these circumstances that the network can become congested.

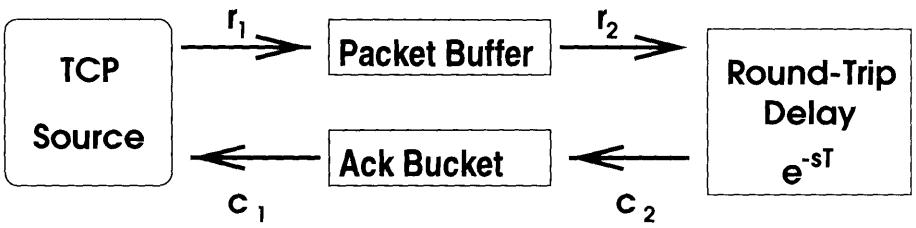

Figure 6-2: System Model

6.3.1 Model

In figure 6-2, we can see a block diagram representation of TCP's dynamics. The TCP source, packet buffer, and ack bucket are different components in the system which communicate with each other through the signals r_1, r_2, c_1, and c_2. Between the buffer and the bucket there is a delay that models the round-trip time for TCP data to reach the destination and for an acknowledgment to return to the ack bucket. Therefore, $c_2(t) = r_2(t - T)$.

The TCP source behavior can be characterized by the following state-space representation:

$$\dot{w}(t) = \begin{cases} c_1(t) & \text{if } w < W_{mid} \\ \frac{c_1(t)}{w} & \text{if } W_{mid} < w < W_{max} \\ 0 & \text{if } w = W_{max} \end{cases} \tag{6.1}$$

$$r_1(t) = c_1(t) + \dot{w}(t) \ . \tag{6.2}$$

The packet buffer is a limited integrator that is drained by the ATM explicit rate (ER). The size of the packet buffer is denoted by s_1. Its state-space description is:

$$\dot{s}_1(t) = \begin{cases} r_1(t) - ER(t) & \text{if } s_1 > 0 \text{ or } r_1(t) > ER(t) \\ 0 & \text{if } s_1 = 0 \text{ and } r_1(t) \leq ER(t) \end{cases} \tag{6.3}$$

$$r_2(t) = \begin{cases} ER(t) & \text{if } s_1 > 0 \\ 0 & \text{if } s_1 = 0 \ . \end{cases} \tag{6.4}$$

Finally, the ack bucket is also a limited integrator that is drained according to the rate translator. The size of the bucket is denoted by s_2. Its state-space representation is:

$$\dot{s}_2(t) = \begin{cases} c_2(t) - c_1(t) & \text{if } s_2 > 0 \text{ or } c_2(t) > c_1(t) \\ 0 & \text{if } s_2 = 0 \text{ and } c_2(t) \leq c_1(t) \end{cases} \tag{6.5}$$

$$c_1(t) = \begin{cases} \frac{ER(t)}{2} & \text{if } s_1 > 0 \text{ and } w(t) < W_{mid} \\ \frac{ER(t)}{1+1/w(t)} & \text{if } s_1 > 0 \text{ and } W_{mid} \leq w(t) < W_{max} \\ ER(t) & \text{if } s_1 > 0 \text{ and } w(t) = W_{max} \\ 0 & \text{if } s_1 = 0 \ . \end{cases} \tag{6.6}$$

6.3.2 Modes of Operation

There are two major modes of operation for TCP over ATM. The ATM link can be either underutilized or saturated. The first case is not of much interest to us because underutilization of the link means that there is no congestion and neither the packet buffer nor the ack bucket is used for any extended period of time. Since the queue or bucket size is practically zero, TCP will perform in the same way regardless of whether it has a bucket or not.

The second case (link saturation) which is of interest to us is much easier to analyze because the flow of data in the ATM link follows exactly the ATM explicit rate. If the explicit rate is a constant C, then $r_2 = c_2 = C$. Depending on the source window size w, the system could be operating in any of the three phases of operation: slow start, congestion avoidance, or saturation. We will describe how TCP (with and without ack holding) behaves in each of the three phases. From now on, we assume that the ATM link is operating at its maximum capacity (ER).

Slow Start

If TCP fills the ATM pipe before its window size reaches W_{mid}, TCP will be operating in the slow start phase while fully utilizing the ATM link. If t_0 is the time when the ATM pipe gets full, the evolution of window size and the various rates in the two different schemes can be solved analytically. In the first scheme, we are not using ack holding, and therefore the ack bucket is bypassed ($c_1 = c_2$). The evolution of the system during the slow start phase is:

No Ack Holding

$$
\begin{aligned}
c_2(t) &= ER \\
\dot{w} &= c_2 = ER \\
w(t) &= w(t_0) + ER \times (t - t_0) \\
r_1(t) &= c_2 + \dot{w} = 2\ ER \\
s_1(t) &= ER \times (t - t_0) \\
s_2(t) &= 0
\end{aligned}
\tag{6.7}
$$

With Ack Holding

$$
\begin{aligned}
c_2(t) &= ER/2 \\
\dot{w} &= c_2 = ER/2 \\
w(t) &= w(t_0) + ER/2 \times (t - t_0) \\
r_1(t) &= c_2 + \dot{w} = ER \\
s_1(t) &= 0 \\
s_2(t) &= ER/2 \times (t - t_0)\ .
\end{aligned}
\tag{6.8}
$$

We can observe how the ack bucket is able to regulate the TCP flow (so that it complies with the ATM ER) without having to make use of the packet buffer ($s_1(t) = 0$). Furthermore, in this phase, the bucket size s_2 with ack holding grows at half the rate of the packet queue s_1 without ack holding.

Congestion Avoidance

If TCP fills the ATM pipe before its window reaches W_{max}, it will eventually reach the congestion avoidance phase while fully utilizing the ATM link. We let t_1 be the time when TCP first has a full pipe *and* has a window w larger than W_{mid}. Again we can solve for the evolution of the window and queue sizes and the various rates:

No Ack Holding

$$
\begin{aligned}
c_2(t) &= ER \\
\dot{w} &= ER/w(t) \\
w(t) &= \sqrt{2ER(t-t_1) + w^2(t_1)} \\
r_1(t) &= ER \times (1 + 1/w(t)) \\
s_1(t) &= w(t) - ER \times T \\
s_2(t) &= 0
\end{aligned}
\tag{6.9}
$$

With Ack Holding

$$
\begin{aligned}
c_2(t) &= ER/(1 + 1/w(t)) \\
\dot{w} &= ER/(w(t) + 1) \\
w(t) &= -1 + \sqrt{2ER(t-t_1) + (1 + w(t_1))^2} \\
r_1(t) &= ER \\
s_1(t) &= 0 \\
s_2(t) &= w(t) - ER \times T \ .
\end{aligned}
\tag{6.10}
$$

Again, we can observe that the size of the ack bucket s_2 when we use ack holding grows slower than the size of the packet queue s_1 when we don't use such a technique.

Window Saturation

Eventually, if there is no packet loss, the window size will reach saturation. Regardless of whether the pipe is full or whether we are using ack holding or not, all of the states of our system will reach steady state. If the pipe is full, the TCP source will send data at the ATM explicit rate. The window size will stop changing and the queue and bucket size will stop growing. During window saturation, the size of the bucket size using ack holding is the same as the queue size when not using ack holding. Nevertheless, ack holding still offers the advantage that one particular ack is much easier to store than one packet.

6.3.3 Discussion

In all the three TCP phases of operation that we have examined, the ack bucket size grows no faster than the corresponding packet queue. Therefore, whenever there is congestion and the ATM pipe is filled, acknowledgment holding never increases the total number of data elements (packets and acknowledgments) that need to be stored in the network. Holding packets will only shift the burden of retention from the buffer to the bucket, decreasing dramatically the physical memory requirements. While a packet might need several Kbytes of memory to be stored, an acknowledgment only needs a few bits (the size of the TCP sequence number).

We can think of a buffer as a network element which increases the round-trip delay seen by the source. In TCP (or other window-based protocols), increasing the round-trip delay has the effect of slowing down the source rate. Therefore, an increase in buffer size will automatically decrease the source's sending rate. However, increasing the round-trip delay to slow down the source can also be achieved by the acknowledgment bucket. In fact, increasing the round-trip delay can be accomplished by slowing down a packet or acknowledgment anywhere in the network. Regulating current TCP flow without changing the source and destination behavior necessarily requires manipulating and changing the round-trip delay. It is to our advantage to control the round-trip delay where it is easiest. An acknowledgment bucket at the edge device seems to be an ideal place.

6.4 Simulation

The performance of our acknowledgment bucket scheme was simulated using the Opnet simulation tool. As seen in figure 6-3, the configuration consists of a TCP Tahoe source, two edge devices, and a TCP destination. Each TCP host is connected to its edge device through a LAN connection with a 10 microsecond delay. The two edge devices are connected by an ATM WAN connection with a 10 millisecond delay. The ATM connection has a fixed capacity of 32 Mbps and the packets are of size 4Kbytes.

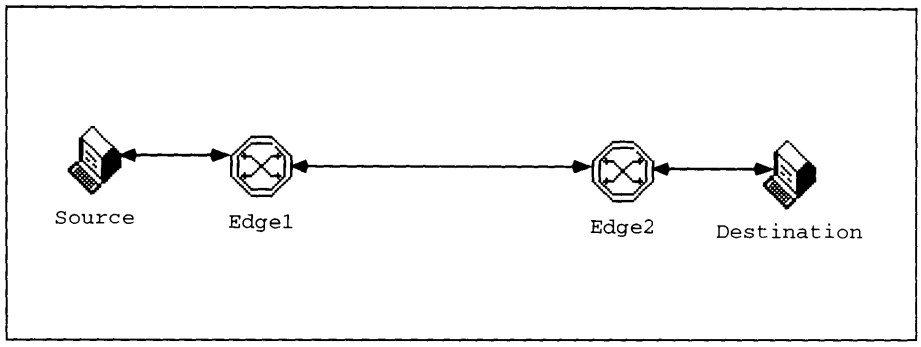

Figure 6-3: Simulation Configuration

Transmission starts with packet number 1. All of the packets arrive safely except packet number 200 which is lost. The destination sends multiple acknowledgments for packet 199, which causes the source to retransmit packet 200. We simulate this scenario with and without using the acknowledgment bucket scheme. The simulations show that by using an acknowledgment bucket, we can obtain the same performance without hardly having to use the forward packet buffer.

In figures 6-4 and 6-5, we can see how the window size at the source evolves with time. The window grows rapidly during the slow start phase, it slows down during the congestion avoidance phase, and eventually stops growing at saturation. In the congestion avoidance phase, we can see how the window grows in the manner described in the analysis section. We can also see how the queue size (in figure 6-4) or the bucket size (referred to as permits in figure 6-5) at the edge follow the change in the window size. When the window size becomes greater than the pipe capacity (round-trip delay times the link rate, about 21 in this case), the difference is unavoidably stored in the packet queue or in the acknowledgment bucket. The growth of the window size is almost identical in both figures. As predicted in the analysis section, the window in the second figure is slightly slower because the withheld acknowledgments slightly decrease the window size.

In figures 6-6 and 6-7, we can see the sequence number of the packets that are leaving the source during the retransmission period. We can see that before the retransmission, the simple scheme (no acknowledgment holding) is ahead of our ac-

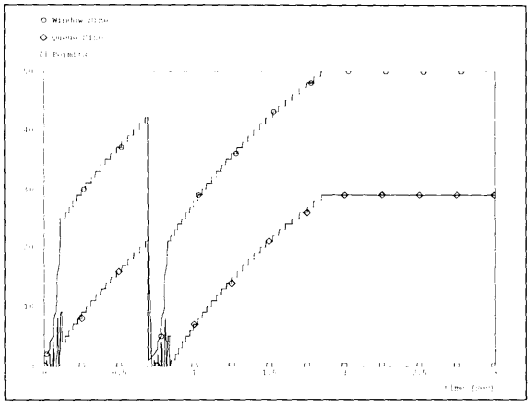

Figure 6-4: Window and Queue Size without Ack Holding

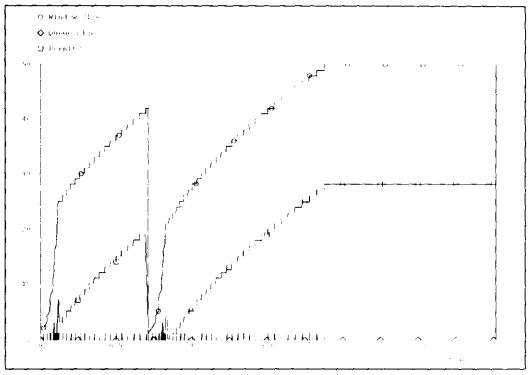

Figure 6-5: Window, Queue, and Bucket Size using Ack Holding

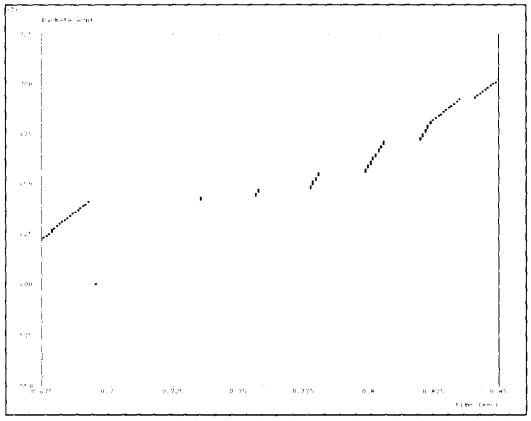

Figure 6-6: Packets Sent by the Source during Retransmit (no Ack Holding)

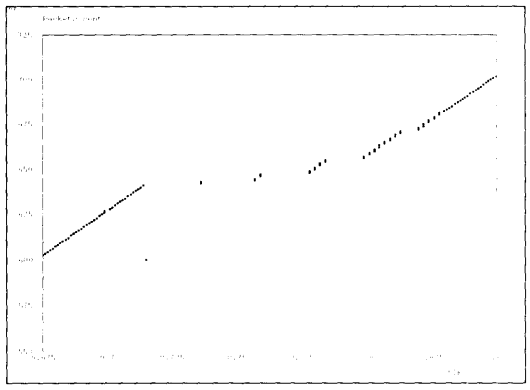

Figure 6-7: Packets Sent by the Source during Retransmit (with Ack Holding)

knowledgment holding scheme in terms of sequence number. However, when a fast retransmit occurs, the retransmitted packet sees a smaller round-trip delay when using acknowledgment holding (there is no forward packet queue). Therefore, when TCP starts increasing its window again, both schemes (with and without acknowledgment holding) are transmitting the same sequence number packets at the same time.

In fact, we can see from figures 6-8 and 6-9 that the output of the edge device is identical in both cases. The lag of the TCP source in the acknowledgment holding case is compensated by a smaller packet buffer. Therefore, adding an acknowledgment bucket is transparent to the ATM network. At any time during the network operation, we can go from case 2 to case 1 (with and without acknowledgment holding) by simply

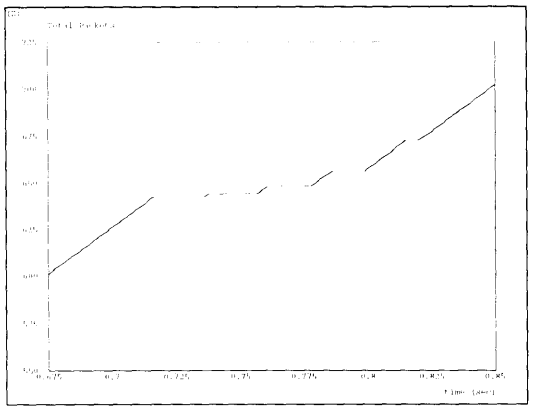

Figure 6-8: Packets Transmitted by the Edge during Retransmit (no Ack Holding)

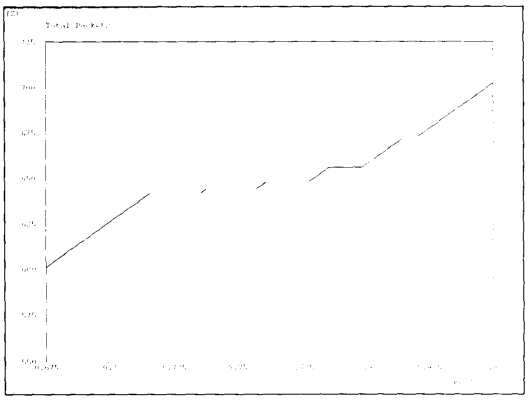

Figure 6-9: Packets Transmitted by the Edge during Retransmit (with Ack Holding)

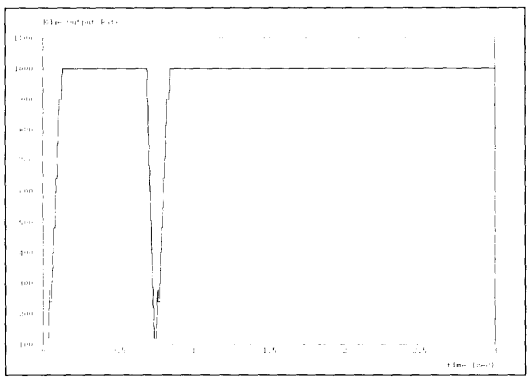

Figure 6-10: Edge Throughput without Ack Holding

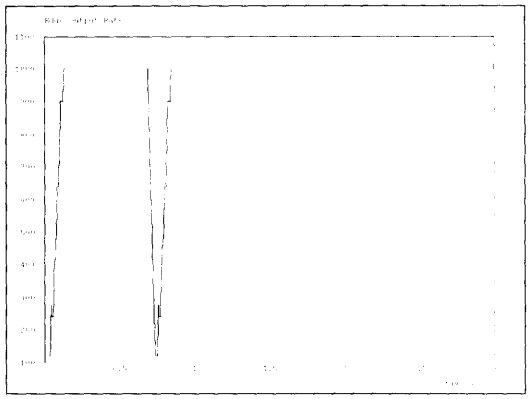

Figure 6-11: Edge Throughput using Ack Holding

releasing all the acknowledgments in the bucket.

Finally, in figures 6-10 and 6-11, we can see the output rate at the edge device in each of the cases. Once again, the throughput is identical.

The only time when the acknowledgment bucket scheme will behave differently at the edge output is when the TCP source timer expires. In this case, the acknowledgment holding scheme will lose the chance of transmitting a number of packets equival to the size of the queue (in the no ack holding scenario) at the time when the lost packet is first transmitted by the source. In TCP Tahoe with fast retransmission this event happens rarely.

Chapter 7

Conclusion

The main goal of this thesis is to simplify the design of network flow control algorithms by developing a unified framework. The earlier part of the thesis introduces a separation principle which divides the controller into two simplified, decoupled, and complementary parts (RRC and QRC). Furthermore, the QRC part (if we use the primal structure) or the RRC part (if we use the dual structure) behaves as an error observer of the other.

With the BRC algorithm, the feedback of the two parts is also separated into two channels. This allows us to implement the QRC observer in the network itself, without having to use artificial delays. We can think of the algorithm as using special information flowing through the network to observe and control the bulk data. We then use the idea of virtual queuing to implement the BRC algorithm on FIFO queues.

In the last chapter, a new technique is introduced to control the flow of TCP over ATM or any networks that can calculate an explicit rate command. Unlike previous proposals, we obtain our results by regulating the circulation of acknowledgments rather than that of packets. This mechanism effectively extends any network-based flow control to all TCP sources. The scheme only requires changes in the ATM edge device, while leaving current TCP and ATM networks untouched.

Future work might be directed toward simplifying the computational complexity of these algorithms. It might be possible to increase the speed of the algorithm at the cost of only minor losses in performance. It might be interesting to investigate

different ways in which to operate the virtual queues which cannot be used in real queues (i.e. allowing them to have negative sizes). Also, work needs to be done on other types of acknowledgment manipulations that could be performed on TCP over ATM as well as on other networks.

Bibliography

[1] Y. Afek et al., "Phantom: A Simple and Effective Flow Control Scheme," Computer Communication Review, vol.26, n.4, 1996.

[2] L. Benmohamed and S. Meerkov, "Feedback Control of Congestion in Packet Switching Networks: The Case of a Single Congested Node," IEEE/ACM Transactions on Networking, Vol. 1, No. 6, 1993.

[3] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall 1992.

[4] D. Cavendish, S. Mascolo, M. Gerla "Rate Based Congestion Control for Multicast ABR Traffic," manuscript to be published.

[5] A. Charny, "An algorithm for Rate Allocation in a Packet-Switching Network with Feedback," Tech. Rep. MIT/TR-601, MIT, 1994.

[6] F. Chiussi, Y. Xia, and V. P. Kumar, "Virtual Queuing Techniques for ABR Service: Improving ABR/VBR Interaction," Proceedings of Infocom '97.

[7] S. Floyd, "TCP and Explicit Congestion Notification," Computer Communication Review, vol.18, no.4, 1994.

[8] M. Hluchyj et. al. "Closed-Loop Rate-based Traffic Management," ATM Forum Contribution 94-0438R2, July 1993.

[9] V. Jacobson, "Congestion Avoidance and Control," Computer Communication Review, vol.18, no.4, 1988.

[10] S. Jagannath and N. Yin, "End-to-End TCP Performance in IP/ATM Internetworks," ATM Forum Contribution 96-1711, December 1996.

[11] R. Jain, S. Kalyanaraman, and R. Viswanathan. "The OSU scheme for congestion avoidance using explicit rate indication,"
ATM Forum Contribution 94-0883, September 1994.

[12] S. Keshav, "Control-Theoretic Approach to Flow Control,"
Computer Communication Review, vol.25, no.1, 1995.

[13] A. Kolarov and G. Ramamurthy, "A Control Theoretic Approach to the Design of Closed Loop Rate Based Flow Control for High Speed ATM Networks,"
Proceedings of Infocom '97.

[14] S. Mascolo, D. Cavendish, and M. Gerla, "ATM Rate Based Congestion Control using a Smith Predictor: an EPRCA Implementation,"
Proceedings of Infocom '96.

[15] M. Mataušek and A. Micić, "A Modified Smith Predictor for Controlling a Process with an Integrator and Long Dead-Time,"
IEEE Transcactions on Automatic Control, vol.41, n.8, 1996.

[16] J. Nagle, "Congestion Control in IP/TCP Internetworks,"
Computer Communication Review, vol.25, no.1, 1995.

[17] H. Ohsaki, M. Murata, H. Suzuki, C. Ikeda, and H. Miyahara,
"Rate-based Congestion Control for ATM Networks,"
Computer Communication Review, vol 25, n.2, 1995.

[18] K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer,"
Computer Communication Review, vol.25, no.1, 1995.

[19] D. Sisalem and H. Schulzrinne, "Switch Mechanisms for the ABR Service: A Comparison Study," TDP '96, La Londes Les Maures, France, 1996, http://www.fokus.gmd.de/step/dor.

[20] D. Sisalem and H. Schulzrinne, "End-to-End Rate Control in ABR,"
WATM '95, Paris, France, 1995, http://www.fokus.gmd.de/step/dor.

[21] D. Sisalem and H. Schulzrinne, "Congestion Control in TCP: Performance of Binary Congestion Notification Enhanced TCP Compared to Reno and Tahoe TCP," ICNP '96, Columbus, Ohio, 1996, http://www.fokus.gmd.de/step/dor.

[22] K. Siu and R. Jain, "A Brief Overview of ATM: Protocol Layers, LAN Emulation, and Traffic Management,"
Computer Communication Review, vol.25, no.2, 1995.

[23] K. Siu and H. Tzeng, "Intelligent Congestion Control for ABR Service in ATM Networks," Computer Communication Review, vol.24, no.5, 1994.

[24] K.-Y. Siu and H.-Y. Tzeng, "Adaptive Proportional Rate Control (APRC) with Intelligent Congestion Indication,"
ATM Forum Contribution 94-0888, September 1994.

[25] O. Smith, "A Controller to Overcome Dead Time,"
ISA Journal, Vol. 6, No. 2, Feb. 1959.

[26] H.-Y. Tzeng and K.-Y. Siu, "Performance of TCP over UBR in ATM with EPD and Virtual Queuing Techniques," Proceedings of Workshop on Transport Layer Protocols over High Speed Networks, IEEE Globecom, Nov. 1996.

[27] Y. Wu, K.-Y. Siu, and W. Ren, "Improved Virtual Queueing and EPD Techniques for TCP over ATM," Technical Report, d'Arbeloff Laboratory for Information Systems and Technology, MIT. January 1997.

[28] N. Yin and S. Jagannath, "End-to-End Traffic Management in IP/ATM Internetworks," ATM Forum Contribution 96-1406, October 1996.