# INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Université d'Ottawa · University of Ottawa

# Sensitivity Methods for Congestion Control in Computer Networks

by

James Aweya, B.Sc. (Hon) and M.Sc.

A thesis submitted to the

School of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

**Doctor of Philosophy**

**in Electrical Engineering**

Ottawa-Carleton Institute for Electrical and Computer Engineering

School of Information Technology and Engineering (Electrical & Computer Engineering)

Faculty of Engineering

University of Ottawa

October, 1999

# Abstract

As computer networks move towards handling diverse traffic types with different service requirements, there is a need for advanced network control mechanisms that can regulate the network traffic to meet the users' service requirements. Service guarantees can be provided if the network makes resource reservations (bandwidth, buffers, priority scheduling, etc.) on behalf of each connection. A problem arises, however, in the control of connections that do not reserve network resources, and hence are not given any performance guarantees by the network. One network control scenario is for these low priority connections to adapt to changing network conditions in order to achieve their data transfer goals and to minimize network congestion. Congestion in this case is a result of a mismatch between the network resources and the amount of traffic admitted for transmission. Consequently, congestion control can be interpreted as the problem of matching the admitted traffic to the network resources. This, in turn, could be viewed as a classical problem of feedback control (i.e., matching the output to the input of a dynamic system). This thesis considers sensitivity methods for the control of congestion in computer networks using neural network models of the system dynamics and system performance sensitivity derivatives. The control methods proposed are used to determine how a feedback-based rate controlled source can satisfy its data transfer requirements by adapting its data rate to changes in the network state.

# List of Publications

[1]. J. Aweya, D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "Multi-Step Neural Predictive Techniques for Congestion Control - Part 1: Prediction and Control Models," To appear, *Int. Jour. Parallel and Distributed Systems and Networks.*

[2]. J. Aweya, D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "Multi-Step Neural Predictive Techniques for Congestion Control - Part 2: Control Procedures," To appear, *Int. Jour. Parallel and Distributed Systems and Networks.*

[3]. J. Aweya, D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "A Gradient-based Binary Feedback Scheme for Congestion Control in Computer Networks," To appear, *Int. Jour. Communication Systems*, Nov. 1998.

[4]. J. Aweya, D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "Optimal Congestion Bit Setting in a Flow Control Scheme using Neural Networks," *Proc. GLOBECOM'98*, Sydney, Australia, November 8 - 12, 1998, pp. 2675 - 2682.

[5]. J. Aweya, D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "Congestion Control using a Multi-Step Neural Predictive Technique," *Proc. GLOBECOM'98*, Sydney, Australia, November 8 - 12, 1998, pp. 1705 - 1714.

[6]. James Aweya and L. Orozco-Barbosa, "Neurocontroller for Buffer Overload Control in a Packet Switch", *IEE Proceedings - Communications*, Vol. 145, No. 4, August 1998, pp. 227 - 233.

[7]. J. Aweya, D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "A Binary Feedback Flow Control Scheme Using Systems Sensitivity Derivatives for Congestion Detection," *Proc. 6th Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'98)*, Montreal, Canada, July 19 - 24, 1998, pp. 112 - 117.

[8]. J. Aweya, Q.J. Zhang and D. Y. Montuno, "Neural Sensitivity Methods for the Optimization of Queueing Systems," (Invited Paper), *World Multiconference on Systemics, Cybernetics and Informatics (SCI'98)*, July 12 - 16, 1998, Orlando, Florida, pp. 638 - 645.

[9]. J. Aweya, Q.J. Zhang and D. Y. Montuno, "Modeling and Control of Dynamic Queues in Computer Networks using Neural Networks," *Int. Conf. on Intelligent Systems and Control (ISC'98)*, Halifax, Canada, June 1-4, 1998, pp. 144 - 151.

[10]. J. Aweya, D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "A Gradient-based Binary Feedback Scheme for Congestion Control in Computer Networks," *Proc. Int. Conf. on Networks and Communication Systems (NCS'98)*, Pittsburg, PA, May 14-16, 1998, pp. 85 - 92.

[11]. J. Aweya, Q. J. Zhang and D. Y. Montuno, "A Direct Adaptive Neural Controller for Flow Control in Computer Networks," *IEEE Int. Joint Conf. on Neural Networks (IJCNN'98)*, Anchorage, Alaska, May 5-9, 1998, pp. 140 - 145.

[12]. J. Aweya, Q. J. Zhang and D. Y. Montuno, "A Neural Network-based Congestion Control Scheme with Fuzzy Parameter Adaptation for Computer Networks," *IEEE Int. Conf. on Fuzzy Systems (IEEE-FUZZ'98)*, Anchorage, Alaska, May 5-9, 1998.

[13]. James Aweya and L. O. Barbosa, "Study of Temporal Behaviour of Packet Loss in Packet Switches with Bursty Traffic Arrivals," *Proc. GLOBECOM'97*, Phoenix, Arizona, Nov. 4 - 8, 1997, pp. 1771 - 1778.

[14]. James Aweya, L. O. Barbosa, and T. Yeap, "A Dynamic Token Allocation Scheme for Flow Control in ATM Multiplexers," *18th Biennial Symposium on Communication*, Kingston, Ontario, June 2 - 5, 1996, pp. 261 - 266.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgements

I would like to express my sincere gratitude to all those who helped me during the course of my doctoral studies. A debt of gratitude is owed to my advisor Prof. Luis Orozco-Barbosa who encouraged me to work in the exciting research area of congestion control, and also for the financial support during my studies at the University of Ottawa. I am also grateful to Prof. Tet Yeap for his encouragement, guidance, and more importantly for his invaluable friendship during the course of my studies. Special thanks also go to Prof. Ahmed Karmouch on whom I could count for help and advice at a moments notice. I wish also to express my thanks to Prof. O. W. Yang for his inspiring and stimulating lectures on computer communication networks which introduced me and helped shaped my understanding of this ever exciting field.

I will like to express my sincere gratitude to my friend and colleague Dr. Delfin Y. Montuno of Nortel Networks for his encouragement and patience, guidance and suggestions, and more importantly for his friendship. His style of research and emphasis on intellectual rigor will always be a guiding influence. I am ever indebted to him. Special thanks also go to Prof. Qi-jun Zhang of Carleton University for providing a stimulating research environment during his sabbatical at Nortel Networks. I am particularly grateful to him for the stimulating discussions on neural networks, fuzzy logic, and optimization. Thanks also go to Michel Ouellette of Nortel Networks for his friendship, advice and all the stimulating discussions on networking. I hope one day his dream of forming Rockland Enterprise will become a reality.

Gratitude is also owed to my friends and colleagues at Dept. 0v10 of Nortel Networks for providing the stimulating atmosphere in which part of this research was done. Special thanks go to my manager Silvana Romagnino and senior manager Kent Felske for their support during the course of this research.

Last, but not the least, my thanks go to Asana, Jessica, and Jeremy for their patience, understanding and support, and for putting up with all the demanding work of this thesis work.

# Chapter 1: Introduction

## 1.1. What is Congestion?

In the design of a packet switched network one faces a multitude of problems, among which are: topology design; capacity allocation; routing procedure; and traffic control procedure [Bert87][Schw87]. In a packet switched network, packets travel from their sources to their destinations "hopping" from one node to another. Once a packet enters an intermediate node, the next node to which it should be sent is selected according to a well-defined decision rule referred to as the routing procedure. The various routing methods, whether fixed or adaptive, local or central, deterministic or stochastic, attempt to direct messages from source to destination such that:

- Delay becomes minimal and
- Traffic is so distributed in the network that congestion is avoided or minimized.

A computer network is a collection of resources to be used by competing users (messages). Resources of such networks can be categorized in two major classes [Pouz76]:

- Basic: buffers, transmission bandwidth, processor time.
- Incidental: name space, table entries, logical channels, etc.

Here, we refer to "users" as the messages to be transmitted. The collection of resources has a limited (finite) capacity which causes conflict to occur between the users of the system. These conflicts can result in a degradation of performance of the system to the point that

the system becomes clogged and the throughput goes to zero; typical behavior of "contention" systems [Agne76]. Even with the best routing procedure, if the network becomes overloaded, this throughput degradation becomes inevitable. Networks cannot afford to accept all the traffic that is offered to them without some control. There must be rules which govern the acceptance of traffic from outside and coordinate the flow inside the network.

Though most networks perform well when lightly used, problems can appear when the network load increases. *Simply*, congestion refers to a loss of network performance when a network is heavily loaded. Since congestion can cause data loss, large delays in data transmission, and a large variance in these delays, controlling or avoiding congestion is a critical problem in network design and operation. This thesis presents some approaches for congestion control in computer networks. In this chapter, we review the state of the art in the control of congestion in (connectionless and connection-oriented) computer networks.

Early research in computer networks led to the development of reservationless (and connectionless) store-and-forward data networks [Bert87][Schw87][Tane81]. These networks are prone to congestion since neither the number of users nor their workload are regulated. Essentially, the efficiency gained by statistical multiplexing and the sharing of network resources is traded off with the possibility of congestion. This problem was recognized quite early [Davi72], and a number of congestion control schemes were proposed; references [Gerl80] and [Pouz81] provide detailed reviews of these earlier studies.

In recent years, there has been a renewed interest in congestion control; a glance at the published literature confirms this trend. The large and rapidly proliferating literature on

congestion control makes it impossible to make a complete list of the bibliography. A number of factors are responsible for this renewed interest. First, the growth in the number of computer networks and the need to tie these networks together has created a very large internetwork (Internet) whose size has made it unmanageable and prone to congestion. The second factor is the development of optical fiber technology. Optical fiber trunks offer data bandwidths that are much higher than traditional copper trunks. These high bandwidth trunks have compounded the problem of congestion control, because of the bandwidth-delay product of these circuits. With such large products, a single source could introduce a data transfer large enough to swamp buffers at the network nodes (i.e., switches and routers), leading to packet losses and excessive end-to-end delays for data transfer.

Despite the large and rapidly proliferating literature on congestion control, there has not been a consensus on the definition of congestion. One definition that seems to be satisfactory is: Congestion is the loss of *utility* to a user due to an increase in the network load [Kesh91a]. Hence, congestion control is defined to be the set of mechanisms that prevent or reduce such a deterioration. Practically speaking, a network can be said to control congestion if it provides each user with mechanisms to specify and obtain utility from the network. For example, if a user desires low queueing delays, then the system should provide a mechanism that allows the user to achieve this objective. If the network is unable to prevent a loss of utility to a user, then it should try to limit the loss to the extent possible, and, further, it should try to be fair to all the affected users. Thus, in reservationless networks (i.e., packet networks in which resources are not explicitly reserved for data sources), where a loss of utility at high loads is unavoidable, the concern is not only with the extent

to which utility is lost, but also the degree to which the loss of utility is fairly distributed to the affected users.

A network can provide utility in one of two ways. First, it can request that each user specify a performance requirement, and can reserve resources so that this level of performance is always available to the user. Some examples are the CBR (Constant Bit Rate) and VBR (Variable Bit Rate) services specified in [ATMF96]. This is proactive or reservation-oriented congestion control. Alternatively, users can be allowed to send data without reserving resources, but with the possibility that, if the network is heavily loaded, they may receive low utility from the network. This second method is applicable in reservationless networks. Examples are the UBR (Unspecified Bit Rate) and to some extent, the ABR (Available Bit Rate) services defined in [ATMF96], and the ARPANET (which is the predecessor to the present day worldwide network, the *Internet*) described in [Carr70][Cerf74][Fran70][Hear70][Kahn71][Robe70]. In this case, users must adapt to changes in the network state, and congestion control refers to ways in which a network can allow users to detect changes in network state, and corresponding mechanisms that adapt the user's data flow to changes in this state.

In a *strict* proactive scheme, the congestion control mechanism is to make reservations of network resources so that resource availability is deterministically guaranteed to admitted data connections. In a reactive scheme, the data sources need to monitor and react to changes in network state to avert congestion. Both styles of control have their strengths and weaknesses. With proactive control, users can be guaranteed that they will not experience loss of utility. On the other hand, to be able to make this guarantee, the number of users has to be restricted or controlled, and this could lead to underutilization and some-

times overload of the network if the admission control mechanism for the data sources is not well designed. Reactive control allows much more flexibility in the allocation of resources. Since users are typically not guaranteed a level of utility by the network, network resources can be dynamically allocated. However, there is always a chance that traffic burst will overload the network, causing performance degradation and hence, congestion.

It is important to realize that proactive and reactive control do coexist in most networks. A network can support, for example, two types of users: guaranteed (or high priority) service users and best effort (or low priority) service users [ATMF96][IETF94]. Guaranteed service users are given a guarantee of quality of service, and resources are reserved for them. In this case, since every user has some reservation, some utility is guaranteed. Best effort service users are not given guarantees and they use up whatever resources are left unutilized by guaranteed service users. At times of heavy network load, users compete for resources kept in a common pool. In addition, a network may reserve some minimum amount of resources for each best effort user [ATMF96][ChenT96].

There are several reasons why congestion arises in computer networks some of which are:

- **Large bandwidth-delay product:** The rate of a communication link multiplied by the round trip time determines the amount of data that a connection must have outstanding in order to fully utilize the network. The round trip time is bounded from below by the speed of light propagation delay through the network. Hence, as the raw trunk bandwidth and the rate of a connection increases, so does the amount of outstanding data per

connection. Since this far exceeds most buffer sizes, data loss is likely, leading to a loss of utility. Thus, a large bandwidth-delay product causes problems for reactive control and can lead to congestion.

- **Speed Mismatch:** If a switch connects a high speed line to a slower line, then a bursty connection can, when sending data at the peak rate, fill up its buffer share, and subsequently lose packets at the switch. This creates congestion for loss-sensitive connections.
- **Topology:** If several input lines simultaneously send data through a switch to a single outgoing line, the outgoing line can be overloaded, leading to large queueing delays, and possible congestion for delay-sensitive traffic. This is a special case of the speed mismatch problem noted above.
- **Misbehavior:** Congestion can be induced by misbehaving sources that transmit unregulated streams of back-to-back packets into the network. Computer network must protect themselves and other sources from such misbehavior.
- **Dynamics:** As network speeds increase, the dynamics of the network also changes. Since queues can build up faster, congestion can be expected to occur much more rapidly, and perhaps have catastrophic effects [Nagl87][Tane81].

## 1.2. Congestion Control

In computer networks where the sources do not make resource reservations, control when needed has to be reactive. A reactive congestion control scheme can be implemented at two locations: at the switches, where congestion occurs, and at the sources, which control

the net inflow of packets into the network. Typically, a switch uses some metric (such as overflow of buffers) to determine the onset of congestion, and implicitly or explicitly communicates this problem to the sources, which reduce their input traffic. A number of problems become apparent in this control scenario:

- **Congestion Detection/Avoidance:** How is congestion to be detected?
- **Congestion Signaling:** How is the congestion problem signalled to the sources?
- **Congestion Recovery:** What actions do the switches take?
- **Selection:** Which sources are held responsible for congestion?
- **Flow Control:** What actions must the sources take?
- **Enforcement:** What if some sources ignore these signals?

These issues must be addressed by every reactive congestion control scheme. Depending upon the choices made in answering each question, a variety of schemes can be obtained. Comprehensive survey [Gerl80][Pouz81][Jain90][Jain96][ChenT96][Arul96] of the literature gives numerous congestion control schemes for connectionless and connection-oriented networks.

### 1.2.1. Congestion Detection/Avoidance

On the issue of congestion detection, one may ask how a switch or source is to detect network congestion. There are several alternatives available, some of which are given below:

- The most common one is to notice that the output buffers at a switch are full, and there is no space for incoming packets. If the switch wishes to avoid packet loss, congestion avoidance steps can be taken when some fraction of the buffers are full. A time average of buffer occupancy can help smooth transient spikes in queue occupancy [Rama88][Rama90].

- A switch may monitor output line usage. It has been found that congestion occurs when trunk usage goes over a threshold (typically 90%) and so this metric can be used as a signal of impending congestion [Irla75][Maji79]. The problem with this metric is that congestion avoidance could keep the output line underutilized, leading to possible inefficiency.

- A source may monitor round-trip delays. An increase in these delays signals an increase in queue sizes, and possible congestion [Jain89].

- A source may probe the network state using some probing scheme (for example, the packet-pair method [Kesh91a]).

- A source can keep a timer that sets off an alarm when a packet is not acknowledged "in time" [Post81]. When the alarm goes off, congestion is suspected [Zhan86].

### 1.2.2. Congestion Signaling

Signaling of congestion information from the congested switch to a source can be implicit or explicit. When signaling is explicit, the switch sends information in packet headers [Rama88] or in control packets such as Source Quench packets [Prue87], choke packets [Maji79], state-exchange packets [Ko91][Sabn89], rate control messages

[ATMF96][Will91], or throttle packets [Kamo81] to the source. Implicit signaling occurs when a source uses probe values [Kesh91a], retransmission timers [Zhan86], throughput monitoring [Wang90], or delay monitoring [Jain89] to indicate the occurrence of congestion.

Explicit congestion signaling imposes an extra burden on the network, since the network needs to transmit more packets than usual, and this may lead to a loss in efficiency if the signaling overhead is not controlled properly. On the other hand, with implicit signaling, a source may not be able to distinguish between congestion and other performance problems, such as a hardware problems [Zhan86].

### 1.2.3. Switch Action

An overloaded switch can signal impending congestion to the sources, and, at worst, can drop packets. The problem is to determine which source to throttle, or whose packets to drop. The fairness of the congestion control scheme depends on this choice made by the switches.

Schemes that rely on line usage as a congestion metric throttle all sources sending packets on an overloaded link [Cher89][Pouz81]. This scheme is unfair in the sense that sources that use a small fraction of the congested link are penalized as severely as large users. The Loss-Load curve scheme [Will91] and the selective DECbit scheme [Rama88] seek to overcome this problem by computing the share of the load due to each user, and selectively dropping packets from that source. These schemes explicitly try to be fair in allocating responsibility for congestion to sources. In the DECbit and Load-Loss schemes, a switch

maintains state information about the demand for bandwidth by each source and its fair share. When the demand exceeds the fair share, and the buffers are getting filled, that source is penalized which is fair by design. Random dropping is an ad hoc technique to achieve fairness and to control congestion. The essential idea is that, when a buffer is full, the packet to drop is selected at random. Hence, users who use more bandwidth, and will probably occupy a larger fraction of the buffer, have a higher chance of packet loss [Mank90].

In some networks, packets carry with them a class indicator and in the case of congestion certain classes are denied entry [ATMF96]. In ATM networks, for example, packets that have their CLP (Cell Loss Priority) bit set are eligible for discard by the switches when congestion occurs in them.

If the buffer usage is a congestion metric, switches drop packets or throttle sources when a source exceeds its share of buffers. This share is determined by a buffer allocation strategy, and the rate at which the buffers are emptied depends upon the service discipline. Thus, the buffer allocation strategy and the service discipline jointly determine which sources are affected. The optimal buffer allocation scheme has received a lot of attention in the literature, and is surveyed in [Gerl80]. Overall, the conclusion in that survey is that a scheme where each output line is guaranteed a minimum number of buffers, and is not allowed to exceed a maximum, is fair and efficient.

Queue service disciplines are implemented in the servers on the output queues of packet switches [Mura90]. The choice of service discipline influences the kind of performance guarantees that can be made to network clients.

### 1.2.4. Flow Control at the Sources

Networks cannot afford to accept all the traffic that is offered to them without some control. There must be rules which govern the acceptance of traffic from outside and coordinate the flow inside the network. These rules are commonly known as flow control procedures. A more precise definition of flow control is given by Pouzin [Pouz76]:

**Definition 1.1:**

> Flow control is the set of mechanisms whereby a flow of data can be maintained within limits compatible with the amount of available resources.

In order to keep the network traffic within desirable limits, flow control, among other things, should be equipped with mechanisms to throttle the use of some resources. A number of schemes have been proposed that operate at the sources. These schemes use the loss of a packet (or receipt of choke information) to reduce the source sending rate in some way. The main throttling tools are:

- Stop-and-go, where a sender continues transmission until it is signalled not to. In a choke scheme, a source shuts down when it detects congestion. After some time, the source is allowed to start up again [Maji79].
- Credit, where the sender receives an indication giving the amount of data it can transmit. One form of control can be accomplished by limiting the number of packets in the network. The isarithmic flow control [Dav72] is a method of global flow control in which the total number of credits in a network are controlled and kept constant. Most end-to-end flow control mechanisms (e.g., the Transmission Control Protocol, TCP) use

a variant of the credit throttling technique and are usually described in terms of a window size [JacoV88], where the number of unacknowledged packets are limited to the "window size". Another credit-based control mechanism is described in [Kung95]. This scheme is similar to a traditional sliding-window data link protocol. An upstream network node cannot send data packets (referred to as cells) to a downstream node until the downstream node gives it credits in the form of credit update cells that specify the amount of available buffer space. Thus, cells sent from the upstream node are always guaranteed to find buffer space in the downstream node, and no cells are lost due to buffer overflow. For fairness, the flow control is applied to each data connection rather than just each link.

- Rate, where the sender transmits traffic at a predetermined rate. In a rate control scheme, when a source detects congestion it reduces the rate at which it sends out packets, either using a window adjustment scheme [Rama90] or a rate adjustment scheme [ClarD87][Kesh91a]. The latter is particularly suitable for sources that do rate based flow control. The advantage of rate control schemes over choke schemes is that rate control allows a gradual transition between sending no packets at all to sending at full blast. Thus, rate control seems to be an attractive alternative for congestion control [ATMF96][ChenT96][Arul96].

### 1.2.5. Enforcement

Some schemes do not support enforcement. For example, in the DECbit scheme [Rama90], if a source chooses to ignore the congestion avoidance bits set by a switch, then the other cooperative sources will automatically give up bandwidth to the ill-behaved

source. The same is true for the TCP scheme [JacoV88]. This problem is avoided partially by the Random-drop scheme [Mank90], and completely by the Loss-load method [Will91] and the ATM Forum traffic management scheme for ABR service [ATMF96]. There is a need for solutions that will work in the presence of ill-behaved sources. That is, a congestion control decision must not only be communicated to the sources, it must also be enforced. The ATM Forum traffic management scheme has this property.

In reservation-oriented networks, networks resources can be allocated at the start of each session. Then, the network can guarantee a (statistical) performance level to a connection by performing admission control. In these networks, it has to be ensured that the sources actually send data at their reserved rates. There are two types of enforcement mechanisms: those that act at the network access point, and those that act at each switch. These schemes seek to monitor and shape user traffic. If some users violate their stated traffic envelope, their data can be dropped or violation-tagged. An efficient way to monitor, and, if necessary, reshape user traffic behavior to make it less bursty, is the leaky-bucket scheme [Ekbe89][Sidi89][Turn86][ATMF96].

Other schemes do enforcement at each switch. The enforcement can be through bandwidth allocation, a queueing discipline and a packet drop policy. One effective mechanism for enforcement is through some form of per-connection queueing discipline [Nagl87].

The queueing discipline controls the order in which packets are sent and the usage of the switch's buffer space, and determines the way in which packets from different sources interact with each other. The queueing mechanism can be thought of as allocating three nearly independent quantities: bandwidth (which packets get transmitted), promptness

(when do those packets get transmitted), and buffer space (which and when packets get discarded by the switch). Currently, the most common queueing mechanism is first-come-first-serve (FCFS). In this scheme, the order of arrival completely determines the bandwidth, promptness, and buffer space allocations.

With simple FCFS switches, a single source sending packets to a switch at a sufficiently high rate can capture an arbitrary high fraction of the bandwidth of the outgoing line. Thus, FCFS queueing is not adequate; more discriminating queueing algorithms (e.g., per-connection queueing) and/or source flow control mechanisms must be used to control congestion effectively in noncooperative environments. There are flow control schemes, such the Available Bit Rate flow control scheme [ATMF96], that can, when implemented with FCFS switches, overcome these limitations and provide reasonable fair and efficient congestion control.

What are the requirements for a queueing algorithm that will allow source flow control algorithms to provide users with adequate utility even in the presence of ill-behaved sources? It has been observed that a queueing algorithm must provide protection, so that ill-behaved sources can only have a limited negative impact on well-behaved sources [Nagl87]. Allocating bandwidth and buffer space in a fair manner, ensures that ill-behaved sources can get no more than their fair share. Nagle [Nagl87] proposed a fair queueing (FQ) algorithm in which switches maintain separate queues for packets from each individual source. The queues are serviced in a round-robin manner. This prevents a source from arbitrarily increasing its share of the bandwidth or the queueing delay received by the other sources. In fact, when a source sends packets too quickly, it merely increases the length of its own queue.

A limited enforcement scheme at the switch is the Random-Dropping (RD) buffer management policy, in which the service order is FCFS, but when the buffer is overloaded, the packet to be dropped is chosen at random [Man90]. This algorithm tends to allocate bandwidth to cooperative sources almost evenly. However, it has been shown that the RD algorithm does not provide fair bandwidth allocation, is vulnerable to ill-behaved sources, and is unable to provide reduced delay to sources using less than their fair share of bandwidth [Floy91][HashE89][Zhan89].

## 1.3. Scope of the Thesis

We have reviewed, in the previous sections, the main ideas and techniques for the control of congestion in computer networks. We defined congestion as the loss of utility to a network client due to an overload in the network. This has motivated the design of various congestion control schemes which allow users to gain utility from the network either by reserving resources to prevent overload, or by reacting to increased network loads. In this thesis, we examine the use of sensitivity methods to determine how a source can satisfy its data transfer requirements by adapting its data rate to changes in network state. Sensitivities are mathematical measures that provide additional insight into the behaviour of a system and they can be used as a basis for inference about input-output relationships in systems. The subject of congestion control is approached from the viewpoint of system sensitivity analysis and adaptive predictive control, which from a review of the existing literature, is a new perspective to congestion control.

The rest of the thesis is organized as follows. In Chapter 2, we present the general concepts of sensitivity analysis and provide information on how neural network models can be used to represent sensitivity functions for the purpose of system control. Chapter 3 describes the basic framework for the control of data traffic into a queueing system using the gradient descent rule and neural sensitivity functions. The objective of the discussion presented in this chapter is to test out the modeling and optimization framework described with a simple control problem. This discussion sets the tone for the other research material presented in the subsequent chapters. The control technique described in Chapter 3 is also a simplified alternative to the one developed in [ChenX92] which describes an adaptive neural congestion control scheme employing two neural network models, one as a neural emulator of the system dynamics, and the other as a neural controller.

Chapter 4 describes a binary feedback congestion control scheme in which the setting of the single bit congestion indication communicated back to the data sources by the network nodes is based upon the sign of the sensitivity of a system performance function. The sign of the performance sensitivity function gives the optimal direction for the data source rate adjustment. In Chapter 5, we present a congestion control technique which employs multi-step neural predictors to account for the control loop delays in the information transfer process in computer networks. Analytical procedures for the source rate control signal computations are described using gradient functions of the neural predictors. Chapter 6 presents the conclusions of the research work and suggestions for further studies.

# Chapter 2: System Sensitivity Analysis

## 2.1. Why Do We Need Sensitivity Analysis?

This chapter introduces the concept of sensitivity and provides basic information on how neural network models can be used to represent sensitivity functions for the purpose of controlling the performance of queueing systems. Sensitivities are mathematical measures that provide additional insight into the behaviour of a system [Bray80][Cruz73][Rada66][Tomo64]. A system designer or a control mechanism should know how the system performance is affected by changes in one or more parameter values. From a practical point of view, it is not sufficient for the design specification to be satisfied for a fixed set of nominal parameter values. Any effect of the system function or any other system characteristic caused by a change in one or more system parameters is referred to as *sensitivity*. A system parameter is called critical if the system sensitivity with respect to this parameter is very large. A small change in such a parameter will strongly affect the system performance.

There are three main reasons for system sensitivity studies:

- Sensitivities help in the understanding of how variations of system parameters influence the system response. By dividing the system parameters into critical and non-critical parameters, an effective method is provided for the purpose of more efficient analysis of the system response.

- They help in comparing the quality of various systems having the same nominal response. Knowledge of the system sensitivity can be used as a basis for comparing different designs.
- They provide response gradients in optimization applications. The objective of optimization is to minimize the discrepancy between the actual and the desired system behaviour.

Queueing systems are inherently complex and hard to analyse, regardless of the modeling framework adopted [Duda83][Fili88][Gros85][Klei75]. This makes it more important to develop new techniques for the analysis, design and, in particular, the control of queueing systems. Generally, the dynamics of queueing systems cannot be easily and adequately modeled through differential (or difference) equations, leaving researchers with inadequate mathematical tools to analyse their dynamic behaviour. Although some models under steady-state have been developed and have enhanced the understanding of queueing systems, the complexity of these models are such that in order to proceed with any manageable analysis of queueing systems (for the purpose of control), simplifying assumptions have to be made. Generally, outside some simple queueing systems at steady-state, analysis becomes extremely complicated [Duda83][Gros85][Klei75]. This complexity has led to some interest in other techniques such as neural networks for the modeling, analysis, and control of queueing systems [ChenX92][IEEE95][IEEE97][Tarr96]. Returning to the issue of whether there is anything at all that can be exploited in neural networks to facilitate their use in the modeling and control of queueing systems, one encouraging possibility lies in the very nature of neural networks as universal approximators of arbitrary functions

[Cybe89][Funa89][Hech89][Horn89][Horn90][Rume88]. Neural networks are capable of modeling arbitrary functions with reasonable degree of accuracy.

For systems that are linear or can be linearized over the required range of operation, conventional linear control theory is adequate for most applications. The assumption of linearity is common, but it does not represent many real-world systems and processes. Most nonlinear systems are linearized over a limited range of operation. In many cases, the assumption of linearity is reasonable and produces good results, especially when a small range of adjustment is involved. Nevertheless, neural networks with their inherent ability to model nonlinear behaviour show advantages that are important in many cases, particularly with complex systems. However, most complex systems are nonlinear and require either a very sophisticated mathematical treatment or simulation. Since the needed parameters for mathematical modeling and simulation models of most complex nonlinear systems are usually not available, an experimental determination of the system characteristics often becomes necessary. This is also where neural networks become very useful. Data from operation or test carried out on the system can be used to train a neural network to emulate the system behaviour, thereby providing a neural model of the system or process.

The analysis in the following sections examines the feasibility of using neural networks and sensitivity analysis to optimize the performance of queueing systems. Sensitivity analysis can be used as a basis for inference about input-output relationships in systems. In addition, a neural network process model and the sensitivity coefficients that can be derived from it, can easily support efficient implementations of gradient-based methods for process optimization. The sensitivity coefficients also identify the system input variables that are most important to improving the system performance. The discussion in the next

sections start with some fundamental results which provide the necessary theoretical foundation for system sensitivity analysis. This is then followed by the application of neural networks in the evaluation of sensitivity coefficients of a system.

## 2.2. Definitions of System Sensitivity

### 2.2.1. Single-parameter Sensitivity

Let us consider the system function $J(\bar{x})$, which is a function of the parameter vector $\bar{x}$ whose components are $x_i$, $i = 1, \ldots, N$. Assuming incremental variations $\delta x_i$ of $x_i$, $i = 1, \ldots, N$, the Taylor-series expansion of the variation $\delta J$ of $J(\bar{x})$ about the nominal parameter vector $\bar{x}_0$ can be written as

$$\delta J = J(\bar{x}) - J(\bar{x}_0) = \sum_{i=1}^{N} \frac{\partial J}{\partial x_i} \delta x_i + \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\partial^2 J}{\partial x_i \partial x_j} \delta x_i \delta x_j + \ldots \tag{2.1}$$

The matrix-vector for (2.1) is given by

$$\delta J = \bar{g}^T \delta \bar{x} + \frac{1}{2} \delta \bar{x}^T \bar{H} \delta \bar{x} + \ldots \tag{2.2}$$

where $\bar{g}$ and $\bar{H}$ are the gradient vector and Hessian matrix, respectively.

Most definitions of system sensitivity are concerned with *infinitesimally small changes* of the vector $x$, i.e., incremental changes that approach zero. We shall first consider defini-

tions of system sensitivity with respect to a single parameter $x$ [Awey98d]. A widely used definition for system sensitivity is the *first-order derivative*

$$S_d(J,x) = \lim_{\delta x \to 0} \frac{\delta J}{\delta x} = \frac{\partial J}{\partial x} . \tag{2.3}$$

This is the simplest sensitivity definition but it is not scale free. The *relative* (or *normalized*) *system sensitivity* is defined as the ratio of a relative, infinitesimally small, change of a system parameter $x$ and the resulting relative change in the system function $J(x)$. One definition is

$$S_n(J,x) = \lim_{\delta x \to 0} \frac{\delta J/J}{\delta x/x} = \frac{x}{J}\frac{\partial J}{\partial x} = \frac{\partial(\ln J)}{\partial(\ln x)} = \frac{x}{J} S_d(J,x) , \tag{2.4}$$

which forms the basis for comparing various designs. When either $x$ or $J$ takes a zero value, the definition (2.4) no longer provides a useful measure, but two other seminormalized sensitivities can be defined:

$$S_{n1}(J,x) = x\frac{\partial J}{\partial x} = \frac{\partial J}{\partial(\ln x)} = xS_d(J,x) \tag{2.5}$$

and

$$S_{n2}(J,x) = \frac{1}{J}\frac{\partial J}{\partial x} = \frac{\partial(\ln J)}{\partial x} = \frac{1}{J} S_d(J,x) . \tag{2.6}$$

The definition (2.5) is used mainly when $J = 0$, while $S_{n2}(J,x)$ is the natural definition when $x = 0$. When both $J$ and $x$ are zero, (2.3) must be used, which is the preferred definition adopted in this thesis.

### 2.2.2. Multiparameter Sensitivity

Ordinary sensitivity provides information regarding system function variation due to single-parameter changes. The function $J$ generally depends on several parameters

$$J = J(\bar{x}) = J(x_1, x_2, \ldots, x_N) \tag{2.7}$$

and it is of interest to consider variation in $J$ when some or all the parameters change *simultaneously*. The change in $J$ due to infinitesimally small changes in all parameters is expressed mathematically by the total differential

$$\delta J = \sum_{i=1}^{N} \frac{\partial J}{\partial x_i} \delta x_i . \tag{2.8}$$

In order to introduce normalized sensitivities into (2.8), divide by $J$ and multiply and divide each term inside the summation by $x_i$:

$$\frac{\delta J}{J} = \sum_{i=1}^{N} \left( \frac{\partial J}{\partial x_i} \cdot \frac{x_i}{J} \right) \cdot \frac{\delta x_i}{x_i} = \sum_{i=1}^{N} S_n(J, x_i) \frac{\delta x_i}{x_i} = \sum_{i=1}^{N} \frac{\partial (\ln J)}{\partial (\ln x_i)} \delta (\ln x_i) . \tag{2.9}$$

Let the set of relative parameter increments $\delta(\ln x_i)$ be represented by the vector

$$\delta \bar{y} = [\delta(\ln x_1)\, \delta(\ln x_2) \ldots \delta(\ln x_N)]^T$$

and the set of gradients $\delta(\ln J) / \delta(\ln x_i)$ by the vector

$$\nabla \bar{G} = \left[ \frac{\partial (\ln J)}{\partial (\ln x_1)} \frac{\partial (\ln J)}{\partial (\ln x_2)} \cdots \frac{\partial (\ln J)}{\partial (\ln x_N)} \right]^T .$$

Then $\delta J / J$ can be expressed as a scalar product

$$\frac{\delta J}{J} = \nabla \bar{G}^T \delta \bar{y} . \tag{2.10}$$

The relative change in the system function $J(x)$ depends on the gradient vector $\nabla \bar{G}$. We can define the *multiparameter sensitivity* of a system as the vector

$$S_m = \nabla \bar{G} . \tag{2.11}$$

Eq. (2.11) is a natural extension of the system sensitivity (2.4) to the multiparameter case.

### 2.2.3. Performance Sensitivity and Queueing Systems

Consider a parameter $x$ that generally affects the behaviour of a queueing system. Thus, the state of the queueing system at time $t$ is written as $X(t, x)$ to indicate this dependence on $x$. When a sample path of this system is observed, its performance is a function of the state history $\{X(t, x)\}$. The notation $L(x, \omega)$ is used to represent the performance as a function of $x$ and of $\omega$, a symbol used to denote the particular sample path that was observed. For any given $x$, since, in general, $\{X(t, x)\}$ is a stochastic process, $L(x, \omega)$ is a random variable. To emphasize the fact that this performance is sample-path dependent, reference [Cass93] refers to $L(x, \omega)$ as a sample path performance function or simply sample function. As an example, consider a single-server queueing system. Let $x$ be the

service rate and $L(x, \omega)$ the server utilization corresponding to a sample path $\omega$. In this case, $\omega$ represents the sequences of interarrival and service times that characterize this sample path [Cass93].

For practical purposes, $L(x, \omega)$ gives an idea of the system's performance, but it only applies to a single sample path; it is entirely possible that the performance observed under a different sample path is drastically different. Thus, the average performance, that is, the expectation over all possible $\omega$ is given by

$$J(x) = E[L(x, \omega)] . \tag{2.12}$$

The sample path performance $L(x, \omega)$ can be thought of as one sample in an effort to estimate the true performance $J(x)$. If $n$ separate sample paths, denoted by $\omega_1, \omega_2, \ldots, \omega_n$ are observed, then an estimate of $J(x)$ is given by

$$\hat{J}(x) = \frac{1}{n} \sum_{i=1}^{n} L(x, \omega_i) . \tag{2.13}$$

If the sample paths observed are independent and yield independent, identically distributed (iid) sequence $\{L(x, \omega_1), \ldots, L(x, \omega_n)\}$, then, by the strong law of large numbers, the following expression is obtained

$$J(x) = \lim_{n \to \infty} \left[ \frac{1}{n} \sum_{i=1}^{n} L(x, \omega_i) \right] . \tag{2.14}$$

In most practical applications, only one sample path $\omega$ is observed from which the performance measure $J(x)$ is estimated, i.e.,

$$\hat{J}(x) = L(x, \omega) . \tag{2.15}$$

A typical problem faced in the design or control of a queueing system is that of determining how performance is affected as a function of some parameter $x$. Given a parameter $x$ and a performance measure $J(x)$, the interest is in the sensitivity of $J$ with respect to $x$, i.e., the derivative $\partial J/\partial x$ evaluated at $x = x_0$. The derivative $\partial J/\partial x$ at $x = x_0$ is known as the sensitivity of $J$ with respect to $x$ at the point $x = x_0$. This provides at least some local information regarding the effect of $x$ on performance. For instance, knowing the sign of the derivative $\partial J/\partial x$ at $x = x_0$ immediately tells the direction in which $x$ should be changed. Also, if $\partial J/\partial x$ is small, it can be concluded that $J$ is not very sensitive to changes in $x$. Thus, attention may be paid on a different parameter (if possible) for improving performance, rather than trying hard at getting minimal improvements from fine-tuning $x$. There are a number of areas where sensitivity information is used for the purpose of analysis and control. A brief discussion of some of these areas is given below [Cass93].

1. Knowing the sign of the derivative $\partial J/\partial x$ at some point $x = x_0$ immediately gives the direction in which $x$ should be changed. The magnitude of $\partial J/\partial x$ also provides useful information in a design or control process. If $\partial J/\partial x$ is small, it can be concluded that $J$ is not very sensitive to changes in $x$ and hence concentration on other parameters may improve performance. In fact in the case of multiple parameters $x_1, x_2, \ldots, x_m$, the sen-

sitivities $\partial J/\partial x_1, \ldots, \partial J/\partial x_m$ can be used to determine the most and least critical parameters, and guide the designer or controller toward the most promising area for improvement.

2. It is often the case that sensitivity analysis provides not just a numerical value for the sample derivative, but an expression which captures the nature of the dependence of a performance measure on the parameter $x$. The simplest such case arises when $\partial J/\partial x$ can be seen to be always positive (or always negative) for any sample path. More generally, the form of $\partial J/\partial x$ can reveal interesting structural properties of a system, e.g., monotonicity, convexity.

3. The sensitivity $\partial J/\partial x$ can be used in conjunction with various optimization algorithms whose function is to gradually adjust $x$ until a point is reached where $J(x)$ is minimized. In practice, the interest is in finding a value of $x$ such that the resulting performance $J(x)$ is satisfactory. If no other constraints on $x$ are imposed, at this point it is expected that $\partial J/\partial x = 0$. A typical algorithm is the gradient descent rule described in discrete-time by

$$x(n+1) = x(n) - \eta(n)\left(\frac{\partial J}{\partial x}\right)_{x = x(n)}, \qquad n = 1, 2, \ldots \tag{2.16}$$

where the parameter is gradually adjusted from the initial value $x_0$ until $\partial J/\partial x = 0$ for some $x(n)$. The amount of adjustment is proportional to the value of the derivative evaluated at $x = x(n)$, with properly selected coefficients $\eta(n)$, $n = 1, 2, \ldots$ (these coefficients are referred to as *step sizes*, *learning rates*, or *scaling factors*).

4. Derivative information including not only first-order, but also higher-order derivatives $\partial^2 J/\partial x^2, \partial^3 J/\partial x^3, \ldots$ can be used for analysis and control purposes.

In addition to the above applications of sensitivity information for control and optimization, sensitivities can also be used for characterizing the statistical distribution of system parameters. Appendix A describes how first and second order sensitivities can be used for characterizing the statistical distribution of system parameters.

## 2.3. Neural Networks and Sensitivity Analysis

Neural networks have many remarkable features, such as the ability to process noisy, sparse, and incomplete data, high fault tolerance which enables the networks to operate properly even with some damaged neurons or weighted connections in the networks, and their ability to respond in real-time due to their inherent parallelism. With this desired features, neural networks have gained very rich applications in pattern recognition, image processing, speech recognition, and adaptive control. For many applications, the training process of a neural network is terminated as soon as the convergent criteria are reached. Then the trained network is used to retrieve the stored information through the recall process. Considerable research on neural networks is focused on developing new learning algorithms, exploring new network architectures, and extending new fields for network applications. Sensitivity analysis is, on the other hand, focused on the study of a trained network itself to obtain useful information other than what can be obtained through conventional recall [Awey98d].

The capability of multilayer feedforward neural networks to approximate an unknown mapping arbitrarily well has been studied in the literature [Cybe89][Funa89][Hech89][Horn89][Horn90][Rume88]. It has been shown in [Horn90] that multilayer feedforward networks with as few as a single hidden layer and an appropriately smooth hidden layer activation function are capable of approximating with arbitrary accuracy an arbitrarily function and its derivatives. It is also noted that the function to be approximated need not be differentiable in the classical sense as long as it possesses a generalized derivative, as in the case for certain piecewise differentiable functions. This fundamental result provides the necessary theoretical foundation for the sensitivity analysis presented in this thesis.

For a continuous function, its derivatives provide very useful information in characterizing the function itself. The first derivative gives the tangential information of the function, and the second derivative gives the curvature information of the function. Sensitivity is usually provided by the first derivative of a function, because it is the ratio of a change in the function's value caused by a change in an independent variable. The larger the absolute value of the first derivative is, the more sensitive the function is with respect to the variable. The sign of the first derivative indicates whether these changes are in the same or opposite directions. The sensitivity of a trained neural network is the analog of that for an ordinary function.

A successfully trained neural network maps an input vector $\bar{X}$ from an $n$-dimensional space to an output vector $\bar{Y}$ in an $m$-dimensional space. This mapping can be expressed as

$$\bar{Y} = f(\bar{X}) , \tag{2.17}$$

where $\bar{Y} = (y_1, y_2, \ldots, y_m)$, and $\bar{X} = (x_1, x_2, \ldots, x_n)$. The first derivative $\partial y_j / \partial x_i$ measures the rate of change in $y_j$ with respect to a change in $x_i$. Hence, it provides information about how sensitive the output $y_j$ is with respect to the input $x_i$. In other words, the larger the absolute value of $\partial y_j / \partial x_i$ is, the more important the $x_i$ is with relation to $y_j$. The input variables $x_i$, $i = 1, 2, \ldots, n$ can then be ranked in the order of importance according to the values of $|\partial y_j / \partial x_i|$. The derivative $\partial y_j / \partial x_i$ can be developed through the structure of the trained neural network.

The objective in this section is to examine techniques for evaluating sample derivatives of the form $\partial y_j / \partial x_i$ using neural networks. Methods for the computation of sensitivity using neural networks fall generally into two classes:

1. Direct methods, by which the first-order derivatives of interest (and any desired higher-order derivatives) are computed directly from the neural network [Guo92][HashS92][Tsou97].
2. Perturbation methods, which involve the introduction of small perturbations at each input of a neural network, one at a time, in both the positive and the negative direction [Cook91][Klim91]. The resultant perturbations at each output in each direction for each input perturbation are measured and averaged.

### 2.3.1. Direct Methods for Sensitivity Analysis using Neural Networks

In this section we describe a direct method for computing the output sensitivities with respect to variations in the inputs for multilayer feedforward neural networks with *differ-*

*entiable nonlinear activation functions*. These sensitivities can be used as a basis for inference about input-output relationships in systems. In addition, a neural network process model can easily support efficient implementations of gradient-based methods for process optimization as will be seen later. An important feature of the technique described here is that the sensitivities are computed one layer at a time, in a systematic manner, starting from the output layer of the multilayer neural network and proceeding backwards toward the input layer. In the analysis, it is assumed that the activation functions are differentiable. For computation convenience, we assume that all the activation functions are of the same form. We also assume that there are no bypassing connections in the network (i.e., direct interconnections can exist only from one layer to the succeeding one). However, while the first assumption about the differentiability of the activation function is fundamental, the latter assumptions are not, and the method discussed here can easily be modified to deal with such deviations.

#### 2.3.1.1. Differentiable Nonlinear Activation Functions

In deriving the sensitivity functions, we assume that the activation functions of the neural network are differentiable and, for example, they can be of the form

$$g(x) = \left(1 + e^{-\alpha x}\right)^{-1}. \tag{2.18}$$

This function is, in fact, the logistic function, one of several sigmoidal functions which monotonically increase from a lower limit (0 or -1) to an upper limit (+1) as $x$ increases. A plot of a logistic function is shown in Figure 2.1a, whose values vary between 0 and 1, with a value of 0.5 when $x$ is zero. An examination of this figure shows that the derivative

(slope) of the curve asymptotically approaches zero as the input $x$ approaches minus infinity and plus infinity, and it reaches a maximum value of $\alpha/4$ when $x$ equals zero as shown in Figure 2.1b. If we take a derivative of (2.18), we get

$$\frac{\partial g(x)}{\partial x} = (-1)\left(1+e^{-\alpha x}\right)^{-2} e^{-\alpha x}(-\alpha) = \alpha e^{-\alpha x} g^2(x) . \tag{2.19}$$

If we solve (2.18) for $e^{-\alpha x}$, substitute it into (2.19), and simplify, we get

$$\frac{\partial g(x)}{\partial x} = \alpha \frac{1-g(x)}{g(x)} g^2(x) = \alpha\,(1-g(x))\,g(x) . \tag{2.20}$$

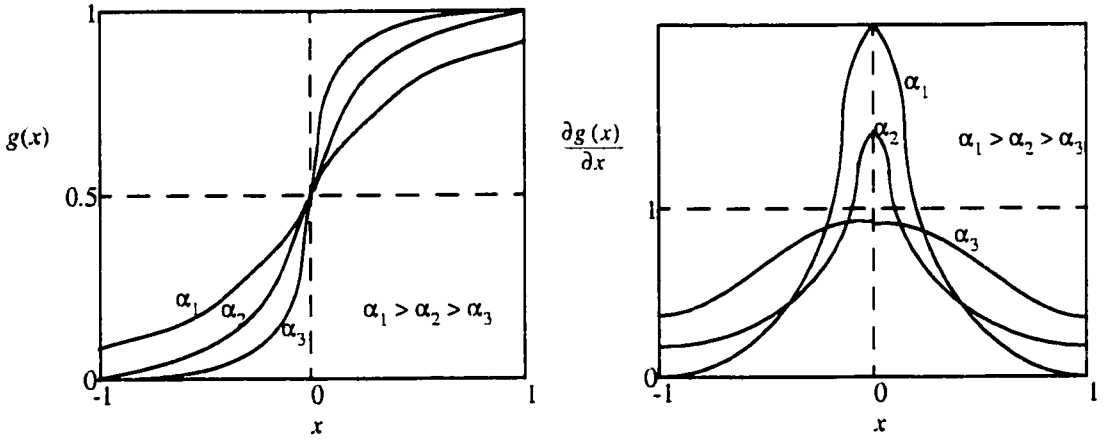



**FIGURE 2.1. Logistic activation function and its first derivative.**

It is important to note that multilayer networks have greater representational power than single-layer networks only if nonlinearities are introduced [Tsou97]. The logistic function provides the needed nonlinearity. However, in the use of the backpropagation algorithm, any nonlinear function can be used if it is everywhere differentiable and monotonically

increasing with $x$. Sigmoidal functions, including logistic, hyperbolic tangent, and arctangent functions, meet these requirements. The arctangent, denoted as $\tan^{-1}$, has the form

$$g(x) = \frac{2}{\pi}\tan^{-1}(\alpha x), \tag{2.21}$$

where the factor $2/\pi$ reduces the amplitude of the arctangent function so that it is restricted to the range -1 to +1. The constant $\alpha$ determines the rate at which the function changes between the limits of -1 and +1 and the slope of the function at the origin is $2\alpha/\pi$. It influences the shape of the arctangent function in the same way that $\alpha$ influences the logistic function in Figure 2.1a. The arctangent function has the same sigmoidal shape as shown in Figure 2.2a. The derivative is

$$\frac{\partial g(x)}{\partial x} = \frac{2}{\pi}\left[\frac{\alpha}{1+\alpha^2 x^2}\right] \tag{2.22}$$

which would be used in place of (2.20) if the arctangent replaced the logistic activation function. The hyperbolic tangent function has the form

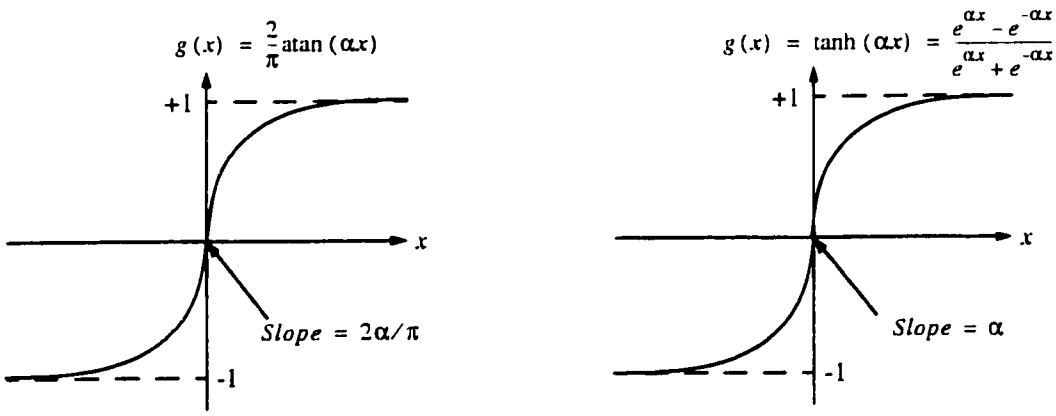
$$g(x) = \tanh(\alpha x) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}} \tag{2.23}$$

and its shape is shown in Figure 2.2b. Its derivative is

$$\frac{\partial g(x)}{\partial x} = \alpha\,\mathrm{sech}^2(\alpha x). \tag{2.24}$$

The slope of the hyperbolic tangent function at the origin is $\alpha$, and it determines the rate at which the function changes between the limits of -1 and +1 in the same general way that $\alpha$ influences the shape of the logistic function in Figure 2.1a.

The use of a sigmoidal function provides a form of "automatic gain control" [Tsou97]; that is, for small values of $x$ near zero, the slope of the input-output curve is steep, producing a high gain, since all sigmoidal activation functions have derivatives with bell shapes of the type shown in Figure 2.1b. As the magnitude of $x$ becomes greater in a positive or negative direction, the gain decreases. Hence, large signals can be accommodated without saturation. This is shown in Figure 2.1a.




a). Arctangent

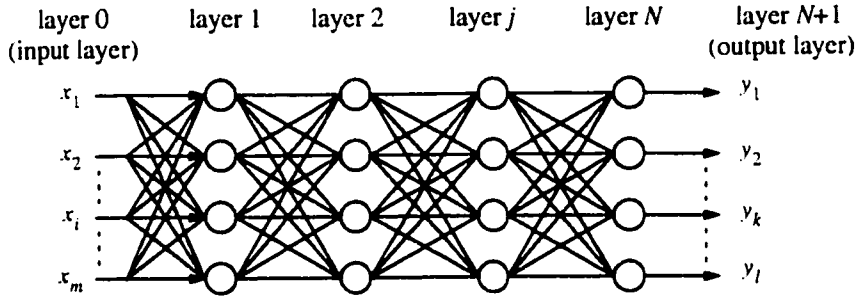
b). Hyperbolic tangent

**FIGURE 2.2. Alternative activation functions.**

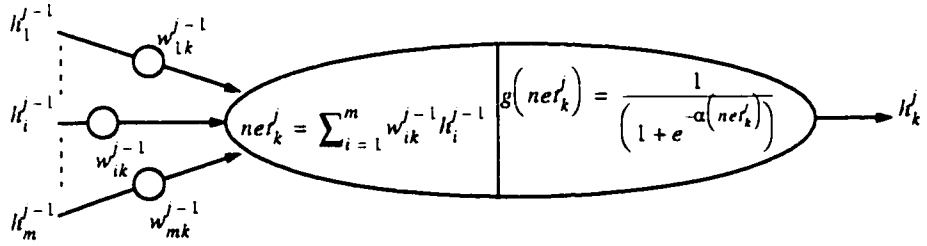
### 2.3.1.2. Neural Architecture and Sensitivity Functions

The feedforward multilayer neural network considered here consists of an input layer, $N$ hidden layers and an output layer (see Figure 2.3). The input vector is introduced to the

input layer (layer 0), which transfers it to the first hidden layer (layer 1). A weighted sum of the input vector is then computed at each neuron in the first hidden layer, based on the weights of the interconnections between the input layer and the first hidden layer. This sum is then used to compute the output of every neuron by applying the logistic sigmoid activation function. Then the neurons in the first hidden layer pass their values to the subsequent layers and finally to the output layer (layer $N+1$). The values of the neurons in the hidden layers and the output layer are calculated in the same manner as in the first hidden layer using the weights of the interconnections between the layers and the logistic sigmoid activation function.




a). Multilayer feedforward neural network.




b). Structure of each neuron.

**FIGURE 2.3. A multilayer feedforward neural network.**

The first order output sensitivities are computed by applying a simple backward chaining partial differentiation rule. First, the output sensitivities with respect to variations in the values of the neurons of layer $N$ are calculated, then backward chaining is employed to calculate the output sensitivities to variations in the neural network input variables. This is done as follows [Guo92][HashS92][Tsou97] using the logistic activation function (2.18),

- For the neurons in layer $N$:

$$\frac{\partial y_k}{\partial h_i^N} = \frac{\partial y_k}{\partial net_k^{N+1}} \cdot \frac{\partial net_k^{N+1}}{\partial h_i^N} = \alpha y_k (1 - y_k) \cdot w_{ik}^N, \qquad \forall i, k. \tag{2.25}$$

- For the neurons in the remaining hidden layers (layers $j$, $j = N-1, \ldots, 1$),

$$\begin{aligned} \frac{\partial y_k}{\partial h_i^j} &= \sum_l \frac{\partial y_k}{\partial h_l^{j+1}} \cdot \frac{\partial h_l^{j+1}}{\partial net_l^{j+1}} \cdot \frac{\partial net_l^{j+1}}{\partial h_i^j} \\ &= \sum_l \frac{\partial y_k}{\partial h_l^{j+1}} \cdot \alpha h_l^{j+1} \left(1 - h_l^{j+1}\right) \cdot w_{il}^j, \qquad \forall i, k. \end{aligned} \tag{2.26}$$

- For the input layer (layer 0):

$$\frac{\partial y_k}{\partial x_i} = \sum_l \frac{\partial y_k}{\partial h_l^1} \cdot \frac{\partial h_l^1}{\partial net_l^1} \cdot \frac{\partial net_l^1}{\partial x_i} = \sum_l \frac{\partial y_k}{\partial h_l^1} \cdot \alpha h_l^1 \left(1 - h_l^1\right) \cdot w_{il}^0, \qquad \forall i, k, \tag{2.27}$$

where $y_k$ is the output of the $k$th neuron in the output layer (layer $N+1$), $h_i^j$ is the output of the $i$th neuron in layer $j$, $j = 1, \ldots, N$, $x_i$ is the $i$th input to the neural network, $net_i^j$ is the weighted sum of the input to the $i$th neuron in the $j$th layer, $j = 1, \ldots, N+1$, and $w_{ik}^j$ is the connection weight between the $i$th neuron in layer $j$ and the $k$th neuron in layer $j+1$,

$j = 0, \ldots, N$. Second order sensitivities, and even higher order ones, of the neural network outputs with respect to the inputs can be calculated using a backward chaining rule similar to that for the first order sensitivities. Appendix B describes in detail how the second order sensitivities are derived. The above equations show that the partial derivative $\partial y_k / \partial x_i$ depends not only on the information learned by the neural network, which is stored distributively in the connections $w_{ik}^j$, but also on the activation functions of the neurons.

### 2.3.2. Perturbation Methods for Sensitivity Analysis using Neural Networks

An experimental evaluation of sensitivity coefficients, sometimes called the "dither" method, is possible after a neural network has been trained [Cook91][Klim91]. It involves the introduction of small perturbations of each input $x_i$, one at a time, of about 0.5% in both the positive and negative directions. The resultant perturbations of each output $y_j$ in each direction for each input perturbation are measured and averaged. The introduction of a small perturbation in one of the input neurons usually produces perturbations in the outputs of all neurons connected directly or indirectly to that input neuron. The ratio of the magnitude of the perturbation in the output of a specific output neuron to the perturbation in the input node is defined as the sensitivity. A perturbation of $x_i$ of the multilayer neural network produces perturbations in all of $y_j$. Hence, the sensitivity for each perturbation $s_{j,i}$ is given by $s_{j,i} = \Delta y_j / \Delta x_i$. The use of this instantaneous approximation often causes

ambiguity for sensitivity evaluation when disturbances are added to the system. Thus, the sensitivity coefficient is taken as

$$\mathcal{S}_{j,i} = \frac{\overline{\Delta y_j}}{\overline{\Delta x_i}}, \tag{2.28}$$

where the averages are taken over the perturbations in the positive and negative directions. If there are $n$ inputs and $m$ outputs, then there are $nm$ sensitivity coefficients that can be evaluated experimentally. The principal problem with this method is that the values are valid only for the particular location in problem space represented by the values of the inputs and outputs before they are perturbed. This method also produces only first-order sensitivities unlike the direct methods which can produce second-order and even higher order sensitivities.

## 2.4. Optimization using Sensitivity Functions

### 2.4.1. The Basic Optimization Process

Optimization is the process of determining the optimal solution of a given design or control problem, i.e., the best possible solution in the sense of some prespecified decision criterion. Optimization plays an important part in nearly all design aspects in engineering, and even in non-engineering areas. Optimization can be considered as a complementary step in the design cycle to satisfy the design specification in the best possible way. Normally, it implies the search for improved solutions of preliminary designs. A preliminary design is the outcome of a synthesis or design process. Usually, such a design does not sat-

isfactorily meet the design specification, necessitating the use of an optimization step for improving the design.

As indicated above, optimization is an iterative process. The preliminary design is analyzed, the result of a system analysis are then compared with the design specification, and any deviation is minimized by properly changing a number of appropriate parameter values. Thus, optimization consists of an iterative sequence of system analyses with updating of system parameter values until the design specification is satisfied within a prescribed accuracy.

The optimization problem in system design can be formulated as follows. Given preliminary system $\Gamma$ with a fixed number of adjustable system parameters, we are required to modify these parameters such that the system behavior approaches the design specification as closely as possible. In optimization, three system quantities play an essential role:

1. We assume that a given function $F$ describes the desired system behavior. The function $F$ represents the design specification.

2. An analytic procedure is used to evaluate the function $F$ of the system $\Gamma$. The computed system function is denoted by $\tilde{F}$ to distinguish it from the desired function $F$. The objective of the optimization process is to vary the adjustable system parameters in such a way that $\tilde{F}$ approaches $F$.

3. The above mentioned adjustable system parameters, which will be referred to as the design or control variables, are denoted by the vector $\bar{x} = [x_1, x_2, \ldots, x_N]^T$.

From the above, it is clear that we need a measure for comparing the actual or computed behavior of a system with the desired behavior. This measure, which is usually called the *cost function* is denoted by $E(\bar{x})$. The problem of optimization is reduced to choosing $\bar{x}$ such that $E(\bar{x})$ is minimized. The main advantage of system design based on an iterative optimization procedure is its flexibility and its suitability for design automation. Despite this advantage, we must remain aware of the difficulties that may arise when optimization algorithms are applied to system design. An unreliable solution is usually the result of an inappropriate formulation of the design problem, a wrong choice of the optimization algorithm or a lack of knowledge of the difficulties that may arise.

In the mathematical treatment of optimization procedures, we deal with the vector of parameter increments

$$\Delta\bar{x} = [\Delta x_1, \Delta x_2, \ldots, \Delta x_N]^T,$$

the sensitivity of the objective function

$$\nabla E(\bar{x}) = \left[\frac{\partial E}{\partial x_1}, \frac{\partial E}{\partial x_2}, \ldots, \frac{\partial E}{\partial x_N}\right]^T,$$

and the Hessian matrix $\bar{H}(\bar{x})$ of second derivatives of $E(\bar{x})$ with respect to $\bar{x}$, that is, an $N$x$N$ symmetric matrix whose $i-j$ entry is $\partial^2 E/\partial x_i \partial x_j$. An $N$-dimensional Taylor-series expansion of the objective function $E(\bar{x})$ can be written as

$$E(\bar{x}+\Delta\bar{x}) = E(\bar{x}) + \Delta E(\bar{x}) \tag{2.29}$$

$$= E(\bar{x}) + \sum_{i=1}^{N} \frac{\partial E(\bar{x})}{\partial x_i}(\delta x_i) + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \frac{\partial^2 E(\bar{x})}{\partial x_i \partial x_j}(\delta x_i \delta x_j) + \ldots$$

$$= E(\bar{x}) + [\nabla E(\bar{x})]^T \Delta\bar{x} + \frac{1}{2}[\Delta\bar{x}]^T \bar{H}(\bar{x})\,[\Delta\bar{x}] + \ldots$$

The above equation is a scalar equation, in which $\Delta\bar{x}$ and $\nabla E(\bar{x})$ are vectors in the $N$-dimensional space, while the last term is a quadratic form of the Hessian matrix $\bar{H}(\bar{x})$ . A necessary condition for a minimum of the objective function $E(\bar{x})$ is

$$\nabla E(\bar{x}) = \frac{\partial E}{\partial \bar{x}} = 0. \tag{2.30}$$

An additional sufficient condition for a minimum is that the Hessian matrix $\bar{H}(\bar{x})$ be positive definite. A quadratic form $Q = \bar{x}^T \bar{H} \bar{x}$ or its associated real matrix $\bar{H}$ is positive definite if for all real $\bar{x}$, except for $\bar{x} = 0$, the quantity $Q$ is real and positive. Solving the optimization problem by solving the system of equations (2.30) is usually a laborious task which involves serious difficulties. The numerous solution methods which have been developed for optimization have the aim of avoiding these difficulties. Sensitivity (or gradient) methods which require the computation of the objective function and its derivatives (using neural network models) are considered in this thesis.

Given the function $E(\bar{x})$ , the value of the gradient $\nabla E(\bar{x})$ depends upon the point $\bar{x}_{nom} \in \Re^N$ (denoting the nominal value of $\bar{x}$) at which $\nabla E(\bar{x})$ is evaluated. If the operat-

ing point is changed from $\bar{x}_{nom}$ to $\bar{x}_{nom} + \Delta\bar{x}$, where $\Delta\bar{x} = \eta \nabla E(\bar{x})$, $\eta \ll 1$ a positive constant, it follows that

$$E(\bar{x}_{nom} + \Delta\bar{x}) \geq E(\bar{x}_{nom}),$$

since

$$E(\bar{x}_{nom} + \Delta\bar{x}) \approx E(\bar{x}_{nom}) + \eta [\nabla E(\bar{x}_{nom})]^T [\nabla E(\bar{x}_{nom})].$$

It is this concept that is used in all gradient methods for the optimization of performance indices in static and dynamic systems. If $f$: $\Re^N \times \Re^+ \rightarrow \Re$, the gradient of $E(\bar{x}, n)$ with respect to $\bar{x}$ is time-varying function and we will consider functions of this type later.

In this thesis, the vector $\nabla E(\bar{x})$ is calculated using the neural sensitivity functions described in Section 2.3. Gradient methods typically employ only first-order derivatives of the objective function $E(\bar{x})$. There are other methods which use second-order derivatives. The commonly used gradient method is the well-known gradient descent rule described by (2.16).

When the objective function has the quadratic form

$$E(\bar{x}) = \sum_{i=1}^{M} [e_i(x)]^2$$

where $M \geq N$, the gradient method can be used to solve the optimization problem. The error functions $e_i(x)$ , $i = 1, \ldots, M$ may be provided with weights $w_i$, if desired. The error function is a measure of the discrepancy between a specified system function $F$ and the actual or computed function $\tilde{F}$. Let us define the $M \geq N$ Jacobian matrix $G = [\phi_{ij}]$ , in which the $i-j$ entry is given by

$$\phi_{ij} = \frac{\partial e_i}{\partial x_j}, \quad i = 1, \ldots, M, \quad j = 1, \ldots, N.$$

The derivatives of $E(\bar{x})$ can then be expressed as

$$\frac{\partial E}{\partial x_j} = 2 \sum_{i=1}^{M} e_i \frac{\partial e_i}{\partial x_j} = 2 \sum_{i=1}^{M} e_i \phi_{ij}$$

or in vector-matrix form:

$$\nabla E(\bar{x}) = 2 \bar{G}^T \bar{e} \tag{2.31}$$

where $\bar{e}$ is the column vector of error functions $e_i$, $i = 1, \ldots, M$. The second-order derivatives are given by

$$\frac{\partial^2 E}{\partial x_j \partial x_k} = 2 \sum_{i=1}^{M} \left( \frac{\partial e_i}{\partial x_j} \frac{\partial e_i}{\partial x_k} + e_i \frac{\partial^2 e_i}{\partial x_j \partial x_k} \right) = 2 \sum_{i=1}^{M} \phi_{ij} \phi_{ik} + 2 \sum_{i=1}^{M} e_i \frac{\partial^2 e_i}{\partial x_j \partial x_k} \tag{2.32}$$

In general, the Hessian matrix $\bar{H}$ is difficult to evaluate with sufficient accuracy. This matrix can easily be obtained from a neural network model using the direct sensitivity

method described in Section 2.3.1 which can be used to evaluate the second-order derivatives.

### 2.4.2. Iterative Procedure for Improving System Performance

The discussion presented here is to take advantage of a neural network's ability to model nonlinearities in a complex system. Since most system conditions are always changing, keeping the system at optimum performance can be achieved using the following iterative procedure:

1. Train a neural network model of the system using on-line data.
2. Carry out a sensitivity analysis to determine which variable(s) are the most critical to the process.
3. Adjust one or more of these variables in the direction indicated by the sensitivity analysis.
4. Wait for the next control interval and repeat the above procedure.

This procedure ensures that the system is always moving towards optimal performance for the operating conditions that exist at the time of the analysis. If the system condition changes or the operating conditions change, it is still possible to move toward optimal performance under the existing conditions. This general framework will be used to control congestion in computer networks, which will be the subject of discussion in the next chapters.

## 2.5. Conclusions

In this chapter we first introduced the concept of sensitivity and provided basic information on how sensitivities can be evaluated from a neural network model of a (nonlinear) system. Computational aspects of sensitivity evaluation were discussed. We have determined that an important piece of information that a system controller should know is how the system performance is affected by changes in one or more system parameter values. Any effect of the system function or any other system characteristics caused by a change in one or more system parameters is referred to as sensitivity. A system parameter is called critical if the system sensitivity with respect to this parameter is very large. A small change in such a parameter will strongly affect the system performance. Our discussion also showed that system sensitivity plays an important part in the design and optimization of reliable systems. The objective of optimization is to minimize the discrepancy between the actual and the desired system behavior. Optimization is usually an iterative process. The main advantage of system design and control based on an iterative optimization procedure is its flexibility and its suitability for design automation. The sensitivity concepts introduced in this chapter will form the basis for the congestion control schemes described in the rest of the thesis.

# Chapter 3: Control of Dynamic Flows in Queueing Systems

## 3.1. Introduction

Simply put, congestion in a queueing system is a result of a mismatch between the network resources and the amount of traffic admitted for transmission. Consequently, congestion control can be interpreted as the problem of matching the admitted traffic to the network resources which in turn, can be viewed as a problem of feedback control in some types of networks (e.g., [Benm93][Bolo90][Kesh91b]). In computer networks, in general, frequent and unpredictably changing traffic patterns is a common feature. Since a changing traffic may cause temporary backlogs of data at the nodes within the network, efficient operation of the network must be maintained through congestion control mechanisms that respond to these backlogs as effectively as possible. Frequent and unpredictable traffic changes produce traffic patterns that are difficult to model accurately, especially if the interactions among traffic patterns caused by various access and network control mechanisms are considered. Instead of imposing a specific stochastic structure on the traffic patterns (e.g., Poisson, Markov Modulated Poisson Process, etc.) and network environment, the approach used here is to assume no specific traffic model, but to apply system sensitivity analysis using neural network modeling to the problem. A number of applications of neural networks to the control of communication networks have been cited in the literature (e.g., [ChenX92][Habi96][IEEE95][IEEE97][Tarr96]). The neural network appeal can be primarily explained by the following reasons:

- Neural networks are essentially adaptive systems able to learn how to perform complex tasks. Detailed description and mathematical understanding about the underlying system to be controlled are not required as a neural network can learn from observations or examples during the course of network operation. This adaptive ability is essential in congestion control where a new environment is continuously encountered.
- Neural network control techniques are believed to be able to overcome many of the difficulties that conventional control techniques suffer when dealing with nonlinear systems or systems with unknown structure. Computer network models are typically nonlinear and may be stochastic either in input and output or behaviour.

Since congestion is a major problem in many queueing systems, the aim of this chapter is to examine how system sensitivities using neural network modeling can be applied to the adaptive control of congestion in queueing systems. The development in this chapter (and also in subsequent ones) is motivated by ideas that are familiar in adaptive control theory. This allows us to make parallels and use insights developed in this area. The theory is based on some heuristics and is not backed up by the type of convergence and stability analysis which is commonly applied in the field of linear adaptive control.

The objective of the discussion presented in this chapter is to test out the modeling and optimization framework described with a simple control problem. This discussion sets the tone for the other research material to be presented later in the thesis. The control technique described here is also a simplified alternative to the one developed in [ChenX92] which describes an adaptive neural congestion control scheme employing two neural network models, one as a neural emulator of the system dynamics, and the other as a neural controller.

The congestion control approach adopted here is as follows: The level of network congestion is monitored through the occupancy or queueing delay $y$ of a network buffer, with the control target being some threshold $r$. Based on the difference between $y$ and $r$, the congestion controller associated with a connection periodically calculates an admission rate and supplies the source with the result of the calculation. In turn, the traffic source adapts its transmission rate to a level not exceeding the rate allowed by the controller. Given this framework, the main question is how the admission rate should be calculated so that the network performance is sufficiently good. We give here the answer to this question for a single source transmitting data into a network node buffer whose service rate changes with time because connections are being set-up and terminated, and because their sources do not maintain constant data rates. This means the available capacity in a transmission link for a single connection is reduced (or increased) by an unspecified disturbance term which represents the capacity assigned (or given up) to traffic from other connections sharing the link.

## 3.2. Basic Congestion Control Model

The congestion control problem considered here consists in adaptively regulating access of external traffic into a queueing system to guarantee the desired performance given in the form of a performance bound. Let $\psi(n)$ define the external traffic arrival rate and $r(n)$ be the required performance bound; $y(n)$ denotes the performance observed from the network at time $n$. $r(n)$ and $y(n)$ can be measured as the queueing delay, queue length or any performance measure of interest. The objective of the control is, to devise an adaptive controller placed at the entrance of the queueing system that will maximize the input traffic to the system within the specified performance bound. The absence of control on traffic load can lead to severe violation of the given per-

formance bound. The actual traffic rate admitted to the network by the controller is described by $\lambda(n)$, with the traffic constraint $0 \le \lambda(n) \le \psi(n)$. We describe the congestion control mechanism by the control variable $u(n)$, defining the portion of the offered traffic $\psi(n)$ that can be admitted to the network, i.e.,

$$\lambda(n) = u(n)\psi(n) . \tag{3.1}$$

The control problem is represented by the diagram given in Figure 3.1.

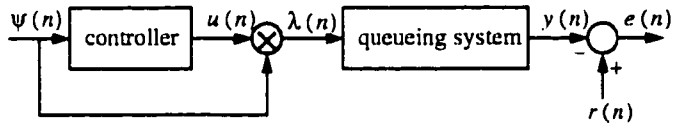



**FIGURE 3.1. General congestion control problem.**

Thus, the control problem in terms of the bounded performance and the traffic constraint can be stated as follows:

$$\begin{array}{ll} \max_{u(n)} & \lambda(n) \\ \text{s.t.} & y(n) \le r(n) \\ & 0 \le u(n) \le 1. \end{array}$$

Reference [ChenX92] describes an adaptive neural control scheme for the above congestion control problem using the so-called indirect control model [Nare90]. This congestion control model employs two neural network models, one as a neural emulator of the queueing system dynamics, and the other as a neural controller. In this chapter, we propose an alternative approach which simplifies the formulation of the control signals and obtain good control performance using a simulation test example described in [ChenX92].

## 3.3. Indirect Adaptive Congestion Control Model

The aim of the neural controller shown in Figure 3.1 is to make control decisions over $\lambda$ based on real-time interaction with the queueing system. Owing to a time-dependent relationship, the control decision $\lambda$ depends on the history of the observed performance and the traffic input rate. Without loss of generality, the $\lambda$ that we have control over takes the form

$$\lambda(n) = G[\psi(n), \lambda(n-1), \ldots, \lambda(n-m), y(n), y(n-1), \ldots, y(n-l)], \tag{3.2}$$

where $\lambda(n-i)$, $i = 1, 2, \ldots, m$ and $y(n-j)$, $j = 0, 1, 2, \ldots, l$ are the time series of controlled traffic input rates and performance observations, respectively. The dimension of the function $G$, $(m+l+2)$, depends on the complexity of the network queueing system considered and on the control accuracy required. Substituting Eq. (3.2) into (3.1). we obtain

$$\begin{aligned} u(n) &= \lambda(n)/\psi(n) \\ &= H[\psi(n), \lambda(n-1), \ldots, \lambda(n-m), y(n), y(n-1), \ldots, y(n-l)] \end{aligned} \tag{3.3}$$

where $\psi(n)$ is absorbed into the function $H$. Since the dynamics of the queueing system (i.e., function $H$) is assumed unknown, a neural network is used to learn the relationship between the control variable $u(n)$ and the system dynamics. The neural network for the controller has $(m+l+2)$ inputs that are the delayed values of relevant signals and one output that can be written as

$$u(n) = \hat{H}[\psi(n), \lambda(n-1), \ldots, \lambda(n-m), y(n), y(n-1), \ldots, y(n-l); W] \tag{3.4}$$

where $W$ represents the adjustable weights of the neural controller. Referring to Figure 3.1, the error signal for the neural controller is

$$e_c(n) = r(n) - y(n) . \tag{3.5}$$

It can be seen from Figure 3.1 that the error backpropagation algorithm cannot be applied directly to the neural controller because of the location of the (queueing) system. The error $e_c(n)$ cannot be propagated through the (queueing) system, and there is no direct learning algorithm available to propagate this error back to the controller. Below we describe three methods that can be used to solve this problem, the simplest and most effective of which is the direct congestion control architecture proposed in this chapter. Before we describe our approach based on a direct control architecture, we give an overview of the perturbation method for error backpropagation, and then the indirect congestion control architecture proposed in [ChenX92].

### 3.3.1. Perturbation Method

As described above, we cannot apply the error backpropagation training for the configuration in Figure 3.1 directly because only the error between the accessible system output $y$ and its desired output $r$ is known to the user. The error $(r - y)$ must therefore be reduced to the input of the queueing system first. The customary error backpropagation rule

$$\Delta W = -\eta \frac{\partial E}{\partial W} \tag{3.6}$$

for weight adjustment within the neural network can be rewritten in the form involving the neural controller output $u$ as follows:

$$\Delta W = -\eta \frac{\partial E}{\partial \lambda(n)} \frac{\partial \lambda(n)}{\partial u(n)} \frac{\partial u(n)}{\partial W} = -\eta \frac{\partial E}{\partial \lambda(n)} \psi(n) \frac{\partial u(n)}{\partial W} , \tag{3.7}$$

where the system output error $E$ can be expressed as

$$E = \frac{1}{2}[r(n) - y(n)]^2. \tag{3.8}$$

To compute the error $E$ reflected at the (queueing) system's input we assume that the system is described with the following algebraic equation:

$$y = y(\lambda). \tag{3.9}$$

The output error $E$ propagates back to the input of the system and $\partial E/\partial\lambda(n)$ can be computed as follows

$$-\frac{\partial E}{\partial\lambda(n)} = [r(n) - y(n)]\frac{\partial y(n)}{\partial\lambda(n)}. \tag{3.10}$$

It can be seen from Eq. (3.10) that the system output error $(r - y)$ propagates back to the system according to the gradient function (or sensitivity coefficient) $\partial y/\partial\lambda$. This describes the performance of the linearized system (3.9) taken at the system's operating point.

If the system equation (3.9) is known, then $\partial y/\partial\lambda$ can be easily produced in an analytical form at the system's operating point. If the system is not known, $\partial y/\partial\lambda$ can be determined by deviating the system's input by the amount $\Delta\lambda$ from its operating point and then measuring the resulting change $\Delta y$ as described in Chapter 2. It involves the introduction of small perturbations of the input $\lambda$ in both the positive and negative directions. The resultant perturbations of the output $y$ in each direction for each input perturbation are measured and averaged. The sensitivity coefficient is then taken as $\overline{\Delta y}/\overline{\Delta\lambda}$, where the averages are taken over the perturbations in the positive

and negative directions. This approach is called the perturbation method and allows for measurement or simulation of approximate values of the gradient $\partial y/\partial \lambda$. The perturbation method yields approximate values only of $\partial y/\partial \lambda$ since it involves finite differences $\Delta y$ and $\Delta \lambda$ rather than infinitesimal differentials. In general, the applicability and usefulness of such measurements for our queueing system is limited. Another problem is that the values are valid only for the particular location in the problem space represented by the values of the input and output before they are perturbed. Next, we describe another approach which overcomes the limitation of the perturbation method by employing a second neural network to emulate the dynamics of the queueing system.

### 3.3.2. Indirect Congestion Control Architecture

In reference [ChenX92], a second neural network is employed to overcome the above difficulty by using the so-called indirect control model [Nare90]. This added neural network is trained to be an emulator of the queueing system. This is to enable the translation of the error signal to the output error of the controller through the neural emulator. This section summarizes the main ideas of the approach used in [ChenX90]. We also extend the discussion in [ChenX92] by describing how to derive some specific gradient functions needed for the control model. The dynamics of the queueing system can be described by an unknown function

$$y(n+1) = f[\lambda(n), \lambda(n-1), \ldots, \lambda(n-m), y(n), y(n-1), \ldots, y(n-l)] . \qquad (3.11)$$

The overall structure of the congestion control scheme is therefore as shown in Figure 3.2.

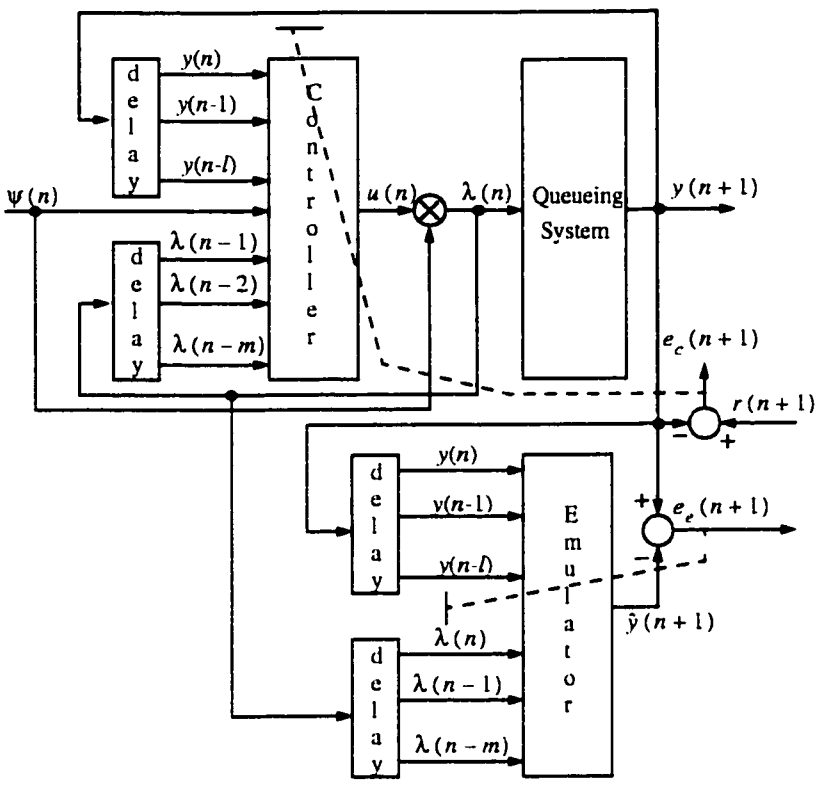



**FIGURE 3.2. Indirect adaptive congestion control scheme.**

The output of the emulator is

$$\hat{y}(n+1) = \hat{f}[\lambda(n), \lambda(n-1), \ldots, \lambda(n-m), y(n), y(n-1), \ldots, y(n-l); V] , \qquad (3.12)$$

where $V$ represents the weights of the neural emulator. The learning of the neural emulator to approximate the function $f$ is a straightforward application of backpropagation neural network, with the following error signal and adaptation of weights:

$$e_e(n) = y(n) - \hat{y}(n), \qquad \Delta V = -\frac{1}{2}\mu \frac{\partial e_e^2(n)}{\partial V} . \qquad (3.13)$$

To compute the gradient for the adjustment of weights $W$ in Eq. (3.5), the chain rule is applied to the error function $e_c^2(n)$, yielding the result

$$\frac{\partial e_c^2(n)}{\partial W} = -2\,[r(n) - y(n)]\,\frac{\partial \hat{y}(n)}{\partial \lambda(n)}\,\psi(n)\,\frac{\partial u(n)}{\partial W} \quad , \tag{3.14}$$

where $\partial y(n)/\partial \lambda(n)$ is replaced by $\partial \hat{y}(n)/\partial \lambda(n)$ as $\hat{y}(n)$ tends to approach $y(n)$ asymptotically. This expression describes the propagation of the error $e_c(n)$ backward through the emulator and down into the controller where the weights $W$ are updated. The neural emulator in a sense translates the error in the final output of the queueing system to the equivalent error in the controller output. $\partial \hat{y}(n)/\partial \lambda(n)$ and $\partial u(n)/\partial W$ are referred to as *backpropagation-to-input* and *backpropagation-to-weights*, respectively. The quantity $\partial \hat{y}(n)/\partial \lambda(n)$ can be calculated by using techniques similar to the error backpropagation algorithm, although some minor modification is needed. If we approximate the system output $y(n)$ by the one-step ahead prediction $\hat{y}(n+1)$, then the gradient $\partial \hat{y}(n+1)/\partial \lambda(n)$ can be determined as explained in Chapter 2 which can be applied to a backpropagation neural network with any number of layers.

It can be seen from Figure 3.2 that there is no feedback loop in the neural emulator as the delayed inputs come from the system (the so-called *series-parallel model*). In contrast, the computation of $\partial u(n)/\partial W$ (i.e., $\partial \lambda(n)/\partial W$) is affected by the past values of $\lambda(n-i)$, $(1 \le i \le m)$ because of the feedback from the output of the neural controller to its input (i.e., a recurrent neural network). For a recurrent backpropagation neural network, modification must be made to evaluate

the gradient $\partial u(n)/\partial W$. It can be verified from variational calculus and Eq. (3.4) that $\partial u(n)/\partial W$ is the solution of the following difference equation:

$$\frac{\partial u(n)}{\partial W} = \frac{\partial \hat{H}}{\partial W} + \sum_{i=1}^{m} \frac{\partial \hat{H}}{\partial \lambda(n-i)} \psi(n-i) \frac{\partial u(n-i)}{\partial W} \quad . \tag{3.15}$$

It should be noticed that $\partial \hat{H}/\partial W$ and $\partial u(n)/\partial W$ differ in that the former treats the inputs as constants. Since $\partial \hat{H}/\partial \lambda(n-i)$ and $\partial u(n-i)/\partial W$ can be computed on-line at every instant of time, the desired gradient can be generated accordingly. Since the first term of Eq. (3.15) reflects the previous basic learning algorithm, the recurrent or dynamic algorithm requires much more computation effort.

In general, the learning control algorithm proceeds in two phases. In the first phase, the emulator is trained using the basic backpropagation learning algorithm. In the second phase, the controller is trained across the composed network. During the second phase only the weights of the controller are changed, and the weights of the emulator are held fixed. The error signal is propagated through the trained or partially trained emulator for the adaptation of the controller weights. The controller is then able to control correctly the traffic input rate by using the emulator as a guide. In practice, it is not necessary that the training of the neural emulator always precedes the training of the neural controller. Identification and control can be done simultaneously in on-line control. As the emulator begins to be trained, the control errors decrease and consequently the controller improves.

## 3.4. Direct Adaptive Congestion Control Model

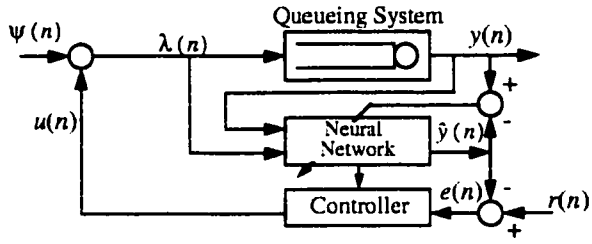



**FIGURE 3.3. Direct adaptive congestion control scheme.**

In this section, we propose an alternative congestion control approach which simplifies the formulation of the control signals [Awey98a][Awey98c][Awey98e]. Figure 3.3 shows the structure of the direct adaptive congestion control scheme. Assume the queueing system can be expressed by Eq. (3.11). The purpose of the control algorithm is to select a control signal $u(n)$, such that the output of the system $y(n)$ is made as close as possible to a pre-specified reference $r(n)$. Using the so-called series-parallel model, the input to the neural network is

$$P = [\lambda(n), \lambda(n-1), \ldots, \lambda(n-m), y(n), y(n-1), \ldots, y(n-l)], \tag{3.16}$$

and the neural model for the unknown system can be expressed by Eq. (3.12). The control signal can be selected such that $\hat{y}(n+1)$ is made as close as possible to $r(n+1)$. A cost function is minimized in order to obtain the optimal control for the next control interval. The performance index is usually of the general form [Sage68]

$$J = \int_{t_0}^{t_f} C\left(\underline{x}(t), \underline{x}^e(t), u(t)\right) dt, \tag{3.17}$$

where $\underline{x}(t)$ is the vector of state variables of the controlled process and $\underline{x}^e(t)$ the desired state. $C$ denotes the cost function, and it can be defined as the distance between the actual system state $\underline{x}(t)$ and the desired state $\underline{x}^e(t)$. $C$ is minimized by selecting the optimal control $u^*(t)$, $t_0 \le t \le t_f$. Since the controlled process is stochastic, $C$ should be considered as an expected value. If we consider directly the minimization of $C$, a large amount of computation is involved for each iteration. The common way to reduce the amount of computation per iteration is to base the control variable updates on the instantaneous performance index

$$J = C\left(\underline{x}(t), \underline{x}^e(t), u(t)\right). \tag{3.18}$$

For our purpose, we define a simple cost function as follows

$$J = \frac{1}{2}e^2(n+1) = \frac{1}{2}[r(n+1) - \hat{y}(n+1)]^2. \tag{3.19}$$

The control signal $u(n)$ should therefore be selected to minimize $J$. A well-known example of a minimization procedure is the gradient descent rule. In the discrete-time case, the control variable is updated according to the rule

$$u(n+1) = u(n) + \Delta u(n) = u(n) - \eta\frac{\partial J}{\partial u(n)}, \tag{3.20}$$

where $\eta > 0$ is the *control learning rate*. It can be seen that the controller relies on the approximation made by the neural network to determine the appropriate control signal. Therefore, it is necessary that $\hat{y}(n+1)$ approaches the real system output $y(n+1)$ asymptotically. This can be

achieved by keeping the neural network training on-line. Differentiating (3.19) with respect to $u(n)$, we obtain

$$\frac{\partial J}{\partial u(n)} = -e(n+1)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)}\frac{\partial \lambda(n)}{\partial u(n)} = -e(n+1)\psi(n)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)}, \tag{3.21}$$

where $\partial \hat{y}(n+1)/\partial \lambda(n)$ is the *dynamic sensitivity derivative*. An algorithm for computing the dynamic sensitivity derivatives $\partial \hat{y}(n+1)/\partial \lambda(n)$ is given below. This algorithm is based on the direct method for sensitivity analysis described in Chapter 2. Substituting (3.21) into (3.20), we have

$$u(n+1) = u(n) + \eta e(n+1)\psi(n)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)}. \tag{3.22}$$

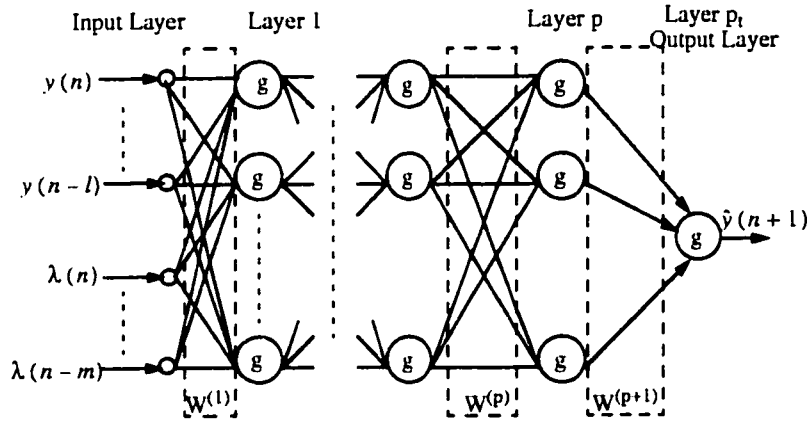
The dynamic sensitivity derivatives are analytically evaluated by using the known neural network structure [Awey98d]. In [Liao93], a different approach is used to estimate the sensitivity of the performance index $\nabla J(n) = \partial J/\partial u(n)$. The approach employs a cross-correlation technique for process sensitivity analysis (i.e., identification of $\nabla J(n)$), where the perturbation signal is a binary random signal.

A control structure based on a neural PID-like control strategy can be developed by redefining (3.19) in the following form

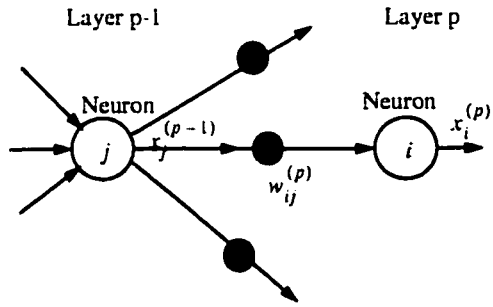
$$J = \frac{1}{2}e^2(n+1) + \frac{1}{2}\beta e^{\,2}(n+1) + \frac{1}{2}\gamma e^{\,2}(n+1), \tag{3.23}$$

where the error and its higher order derivatives are given by $e(n+1) = r(n+1) - \hat{y}(n+1)$, $e\ (n+1) = e(n+1) - e(n)$, $e\ (n+1) = e\ (n+1) - e\ (n)$, and $\beta, \gamma$ are weighting factors. This development is given in Appendix C.

### 3.4.1. The Dynamic Sensitivity Derivatives




a). Feedforward neural model for deriving $\partial\hat{y}(n+1)/\partial\lambda(n)$.




b). Interconnection of neurons.

**FIGURE 3.4. Neural network for computing the dynamic sensitivity derivatives.**

A key requirement for deriving the control algorithm is the determination of the dynamic sensitivity derivatives $\partial \hat{y}(n+1)/\partial \lambda(n)$. Suppose the feedforward neural network used to model the system dynamics is shown in Figure 3.4. Let us define the following parameters: Total number of layers, $p_t$; Layers are numbered, $p = 1, 2, \ldots, p_t$; Neurons are numbered, $i = 1, 2, \ldots, m_{(p)}$; Input layer, $\bar{P} = X^{(0)}$, where $\bar{P}(n) = [y(n), \ldots, y(n-l), \lambda(n), \ldots, \lambda(n-m)]^T$; Output layer, $\hat{y}(n+1) = X^{(p_t)}$. For each neuron $x_i^{(p)} = g\left(z_i^{(p)}\right)$, where $z_i^{(p)} = \sum_{j=1}^{m_{(p-1)}} w_{ij}^{(p)} x_j^{(p-1)}$ and $g(x) = \left(1 - e^{-x}\right)/\left(1 + e^{-x}\right)$, for example. The dynamic sensitivity derivative $\partial \hat{y}(n+1)/\partial \lambda(n)$ is computed using the algorithm in Figure 3.5 which can be applied to a feedforward neural network with any number of layers [Awey98c].

Algorithm for computing $\partial \hat{y}(n+1)/\partial \lambda(n)$ :

Define local gradient for layer $p$, neuron $i$:

$$\delta_i^{(p)} = \begin{cases} g\left(z_i^{(p)}\right), & \text{if } p = p_t \\ \sum_{j=1}^{m_{(p+1)}} \left(\delta_j^{(p+1)} w_{ji}^{(p+1)}\right) g\left(z_i^{(p)}\right), & \text{if } 1 \le p < p_t, \end{cases}$$

where the input layer $X^{(0)} = P(n)$ .

Computation Steps:

Step 1: Compute $\delta_j^{(p)}$ , $j = 1, 2, \ldots, m_{(p)}$ , starting from $p = p_t$ , then backpropagate in neural network recursively to obtain $\delta_j^{(p)}$ for $p = p_t - 1, p_t - 2, \ldots, 2, 1$ .

Step 2: Suppose $\lambda(n) = x_i^{(0)}$ (i.e., element $i = l+2$ of $X^{(0)}$ ), then

$$\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)} = \sum_{j=1}^{m_{(1)}} \delta_j^{(1)} w_{ji}^{(1)} .$$

**FIGURE 3.5. Algorithm for computing the dynamic sensitivity derivatives.**

## 3.4.2. Heuristics for Control Step Size Adaptation

The gradient descent procedure has some well known drawbacks. The first, and simplest one, is just the difficulty in the choice of the value of $\eta$ . Too small a value will result in a very slow convergence, while if the value is too large, the optimization process will diverge, instead of converging to a minimum of the cost function. In the presence of a new control problem, the best value for this parameter is not known, and in fact it will often vary during the course of the optimization. Some heuristics for improving the control performance of the gradient descent method are

proposed below [Awey98b][Awey98c][Awey98e] based upon the observations given in [JacoV88].

**Heuristic 3.1:**

When the derivative of the cost function $\nabla J(n)$ with respect to a control has the same algebraic sign for *several consecutive iterations* of the algorithm, the control rate parameter should be increased.

**Heuristic 3.2:**

When the algebraic sign of $\nabla J(n)$ alternates for *several consecutive iterations* of the algorithm, the control rate parameter should be decreased.

The intent of the heuristics stated here is to ease the choice of the parameter $\eta$, so that with a reasonably chosen starting value, and with periodic adaptation of this parameter, effective control can be attained. There exist several possible implementations of the heuristics described above. Here, we consider only two approaches. Another implementation using fuzzy logic control is given in [Awey98b].

### 3.4.2.1. Exponentially Weighted Sum Approach

The exponentially weighted sum approach implements the heuristics through the addition of a new term to the control update equation [Awey98a][Awey98e]. At time step $n$, the control is updated according to the following rule:

$$\Delta u(n) = -(1-\alpha)\xi\nabla J(n) + \alpha\Delta u(n-1), \tag{3.24}$$

where $\eta = (1-\alpha)\xi$ is the control learning rate, $\xi \in (0, 1]$ is a scaling factor, and $\alpha$ is the exponential weighting factor that determines the relative contribution of the current and past derivatives $\nabla J$ to the current control change. Using the exponentially weighted sum method, the current control update can be expressed as

$$\Delta u(n) = -\eta \sum_{i=0}^{n} \alpha^i \nabla J(n-i) . \tag{3.25}$$

This contribution is the exponentially weighted sum of the current and past derivatives $\nabla J$ where $\alpha$ is the base and the time from the current time step is the exponent. The exponentially weighted sum method corresponds in fact to a recursive low-pass filtering of the gradient of the cost function (see Figure 3.6). It has the effect of attenuating "high frequency" oscillations in the control updates, and amplifying the "DC" components. A large value of $\eta$ may cause oscillations in the control updates. To dampen these oscillations down, the exponentially weighted term with coefficient $\alpha$ can be used. Broadly, $\eta$ acts as a "gain" coefficient and $\alpha$ as a "damping" coefficient which has the effect of damping oscillations in the control updates.

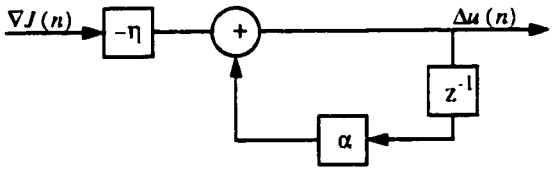



**FIGURE 3.6. Exponentially weighted sum approach for control step size adaptation.**

**Proposition 3.1:**

The exponentially weighted sum approach is an implementation of the heuristics.

*Proof*: When consecutive derivatives $\nabla J$ possess the same sign, the exponentially weighted sum grows large in magnitude and the control is adjusted by a large amount. Similarly, when consecutive derivatives $\nabla J$ possess opposite signs, this sum becomes small in magnitude and the control is adjusted by a small amount.

$\Delta\Delta$

One feature of the exponentially weighted sum method is the sensitivity of the control performance to the numerical values chosen for $\eta$ and $\alpha$. As with the basic steepest descent rule, the best values for these parameters are usually not known for a control problem. A wrong choice of these parameters can result in poor system performance, so an optimization scheme is required that employs continuously adaptive coefficients, which is the next subject of discussion.

### 3.4.2.2. Exponential Increase/Decrease Approach

In this approach we consider a direct implementation of the heuristics [Awey98c][Awey98e]. To simplify the discussions and to illustrate how this approach works, we consider the following simplified implementation. If in two successive iterations the gradient values $\nabla J$ have opposite signs, that means that we have jumped over a minimum and that the control step size is too large. On the other hand, if two successive signs are equal, it appears that the control rate could have moved somewhat faster, while still not passing the minimum. The basic heuristic for adaptation is then simply to decrease the control step size if two successive gradients have opposite signs, and to increase it if they have the same sign. In practice, *it may be appropriate to carry out the adaptation periodically after a number of control steps have elapsed* as stated in Heuristic 1 and 2. A tracking mechanism such as the one described below can be used for this purpose [Awey98b].

Consider, for example, if $\nabla J(n-1) \leq \nabla J(n)$ and $\nabla J(n) > \nabla J(n+1)$, then the change in gradient $CG(n) = \nabla J(n) - \nabla J(n-1) \geq 0$ and $CG(n+1) = \nabla J(n+1) - \nabla J(n) < 0$. This means there has been a sign change from iteration $(n-1)$ to iteration $(n+1)$ of the algorithm. Let us therefore introduce the sign change parameter,

$$SC(n) = 1 - \left| \frac{1}{2} [\operatorname{sgn}(CG(n-1)) + \operatorname{sgn}(CG(n))] \right|, \tag{3.26}$$

where the hard limiter $\operatorname{sgn}(x) = 1$, if $x \geq 0$, and -1 otherwise. The factor 1/2 is to ensure $CG$ is either 0 (no sign change) or 1 (one sign change). The cumulative sum of $CG$ (or $CSC$) thus reflect the history of the sign changes, i.e.,

$$CSC(n) = SC(n) + SC(n-1) + SC(n-2) + \ldots \tag{3.27}$$

The bigger the CSC, the more frequent the sign changes have occurred. For example, for a ten step tracking of the sign changes, we define

$$CSC(n) = \sum_{k=n-9}^{n} SC(k). \tag{3.28}$$

Tracking of the sign change over a number of consecutive iterations, simplifies the control scheme since the control step size is not expected to vary significantly from one control step to the other. We propose using an exponential increase and an exponential decrease of the control step size. More specifically, for a simple *one step adaptation* example, we will update the control step size according to

$$\eta(n+1) = h\eta(n) \text{ if } \partial J/\partial u(n) \text{ and } \partial J/\partial u(n-1) \text{ have the same sign,} \tag{3.29}$$

$$\eta(n+1) = d\eta(n) \text{ if } \partial J/\partial u(n) \text{ and } \partial J/\partial u(n-1) \text{ have opposite signs,} \tag{3.30}$$

where $h$ and $d$ are factors slightly above and below 1, respectively (e.g., $h = 1.2$ and $d = 0.7$). The update of the control parameter can then be performed by

$$u(n+1) = u(n) - \eta(n+1)\frac{\partial J}{\partial u(n)}. \tag{3.31}$$

In the situation in which the optimal value of the control step size were constant throughout the minimization process, the step size would exponentially increase from its initial value, being multiplied by $h$ in each iteration (or, alternately, it would continually decrease, being multiplied by $d$ in each iteration), until it would just pass its optimal value. After that, it would keep in a small oscillation around the optimal value, being multiplied by $d$ when above that value, and by $h$ when below it. In real situations, each control step size will normally have periods of exponential increase or decrease, interspersed with periods of oscillation, as the optimization process progresses.

The choice of the values of the factors $h$ and $d$ for the adaptation must result from a compromise between a fast adaptation (which would require values much above and below 1, respectively) and a small oscillation around the optimal step sizes (which would demand values as close to 1 as possible). In practice, we have found that values close to 1 ($h$ in the range 1.1 - 1.3 and $d$ in the range 0.7 - 0.9, approximately) generally yield good results, and that the exact values that are chosen within these ranges are not crucial in any way, as long as $d \approx 1/h$.

Even with an adaptive control step size, the use of the exponentially weighted term will still be of advantage [Awey98e]. The control filtering can be applied to the gradient component before multiplying by the adaptive step size. The control update equation (3.31) can be replaced by

$$u(n+1) = u(n) - \eta(n+1)\, v(n), \tag{3.32}$$

where

$$v(n) = \frac{\partial J}{\partial u(n)} + \alpha v(n-1), \tag{3.33}$$

$\alpha$ being the exponential weighting parameter and $v(0) = 0$. The exponential increase or decrease of the control step size can, in some situations, lead to overflow or underflow situations. To prevent this, upper and lower bounds to the control step size can be set. These bounds can be, respectively, a few orders of magnitude below the largest representable number, and a few orders of magnitude above the smallest representable positive number, for the system under consideration.

## 3.5. Numerical Results

In this section, a dynamic queueing model taken from [Agne76] and [Fili88] is used to test the performance of the proposed congestion control technique. Similar to the reasons given in [ChenX92], this model with known dynamics is chosen so that the performance of the control technique can be easily checked. Only a first-order difference approximation for this model is assumed, which is aimed at the development of a model that can be conveniently used for evaluating the performance of the control system. This is done to simplify the model presentation and to focus on the congestion control mechanism. For the system model, we consider an M/D/1 dynamic queueing model which is governed by the difference equation $y(n+1) = f[y(n)] + \lambda(n)$, where the function $f$ has the form

$f[y(n)] = \sqrt{[y^2(n) + 1]} - 1$, which is assumed to be unknown, and $y(n)$ and $\lambda(n)$ represent the time-dependent average queueing delay and arrival rate, respectively. The time units are normalized such that the queue service capacity is equal to one. The objective of the control is to regulate the queue arrival rate $\lambda(n)$ subject to the queueing delay bound $r(n)$ specified. The external input rate to the queue is assumed to be $\psi(n) = 0.6 + 0.2\sin(\pi n/20) + 0.1\sin(\pi n/100)$ and the desired queueing delay bound $r(n) = 1.5 + 0.5\sin(\pi n/250)$. In an actual system, in place of the sample-path values obtained from the above queueing model, the quantities $\psi(n)$, $\lambda(n)$ and $y(n)$ will represent real-time measured (or sample) values taken at every control interval. The system models and control algorithms can then be derived from these real-time measurements. In the control scheme, on-line learning is necessary because the training pattern for the neural network changes with time. The on-line control and neural network training algorithm for the control scheme is given in Figure 3.7. In this chapter, *we adopt a one-step adaptation of the control step size*, although in practice it may be appropriate to carry out the adaptation periodically after a number of control steps have elapsed.

At time instant $n$:

Step 1: Get measurement of the system input and output: $\lambda(n)$ , $y(n)$

Get output reference: $r(n+1)$

Available system history: $\lambda(n-1), \ldots, \lambda(n-m)$ , $y(n-1), \ldots, y(n-l)$ .

Neural network history: $\hat{y}(n), \delta_i^{(p)}(n-1), x_i^{(p)}(n-1)$ , for all layers $p$ and all nodes $i$.

Step 2: Neural network feedforward to compute $\hat{y}(n+1)$ .

Neural network backpropagation to compute $\partial\hat{y}(n+1)/\partial\lambda(n)$ .

Step 3: Control and adaptation:

Compute gradient function $\nabla J(n) = \partial J/\partial u(n)$ .

Adapt control learning rate parameter $\eta$ (if necessary).

Compute control input $u(n+1)$.

Step 4: Using $[\hat{y}(n) - y(n)]$ as neural network error to update neural network weights:

$$\Delta w_{ij}^{(p)}(n) = -\beta[\hat{y}(n) - y(n)]\,\delta_i^{(p)}(n-1)\,x_j^{(p-1)}(n-1) + \gamma\Delta w_{ij}^{(p)}(n-1) ,$$

$$w_{ij}^{(p)} \Leftarrow w_{ij}^{(p)} + \Delta w_{ij}^{(p)}(n) ,$$

for $p = p_t, p_{t-1}, \ldots, 1$ , $i = 1, 2, \ldots, m_{(p)}$ , $j = 1, 2, \ldots, m_{(p-1)}$ .

$\beta$ - Neural network learning rate; $\gamma$ - Neural network momentum constant

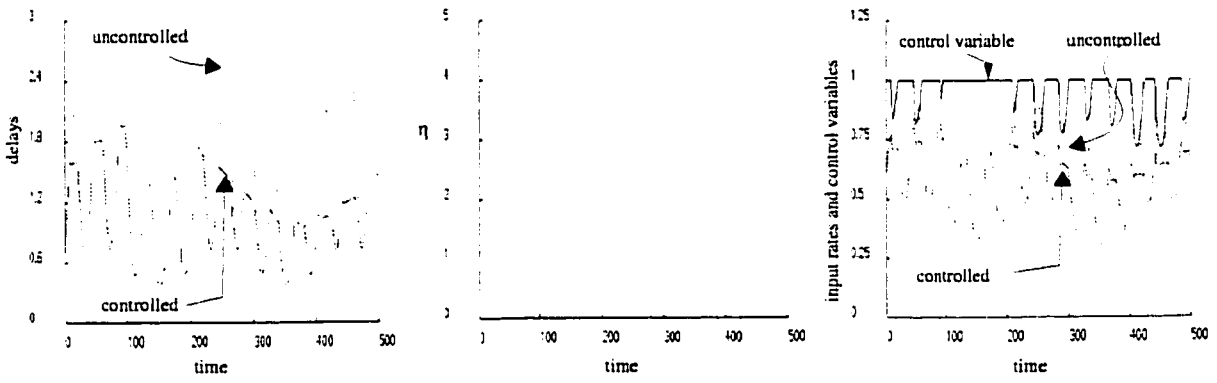
Step 5: Next interval, $n \Leftarrow n+1$ , go to Step 1.

Note: Steps 3 and 4 can be done in parallel.

**FIGURE 3.7. Algorithm for on-line congestion control and neural network training.**

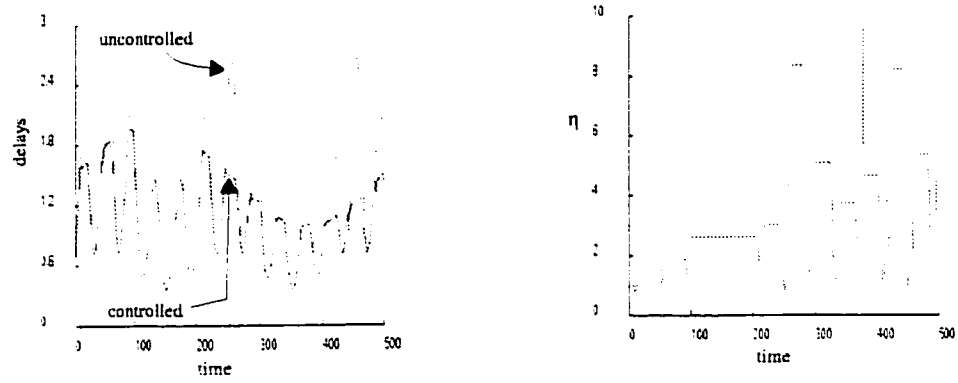
Although the search for the right control parameter can be done off-line through a trial and error adjustment of the parameter, this process is usually slow and is not suitable for real time operation. Through a trial and error adjustment, good control parameters were obtained for the simulation example as shown in Figure 3.8. In fact the chosen value will often vary during the operation of the system when it is used as the initial value for the control optimization process as shown in Figure 3.9. Figures 3.10 and 3.11 show that the exponential increase/decrease adaptation of the control step size with values of $h$ and $d$ close to 1 ($h$ in the range 1.1 - 1.3 and $d$ in the range 0.7 - 0.9, approximately) yield good results even though the initial control step sizes were not "opti-

mally" chosen. The cases of $\eta = 0.01, \alpha = 0.05$ ($\eta = 5, \alpha = 0.05$) result in weak (strong) control performance. The choice of the values of the factors $h$ and $d$ for the control step size adaptation must result from a compromise between a fast adaptation and a small oscillation around the optimal step sizes. As seen in Figures 3.10 and 3.11, increase/decrease parameters of $h = 1.3, d = 0.7$ give fast control step adaptation that results in the desired system performance being met in a smaller number of iterations. In Figure 3.10 with $\eta = 0.01, \alpha = 0.05$, for instance, when $h = 1.3, d = 0.7$, it takes approximately 50 iterations for the system delay specifications to be met, while for $h = 1.1, d = 0.9$ it takes about 250 iterations. The main benefits of the control step size adaptation procedure is that the system eventually converges to optimal performance even if the initial choice of the step size is not good, and also, the step size is continuously adjusted when a new environment is encountered.



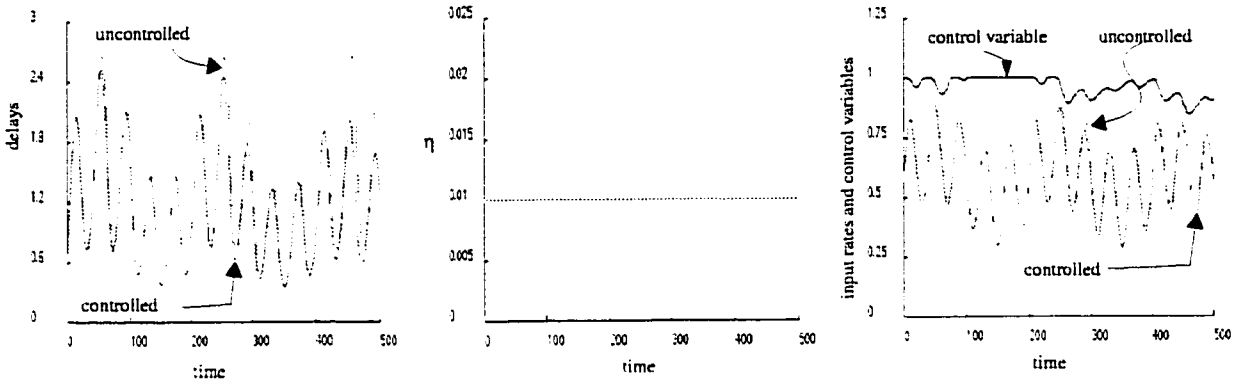

($\eta = 1, \alpha = 0.05, h = 1, d = 1$)

**FIGURE 3.8. Good control parameters found by trial and error and no control step size adaptation.**

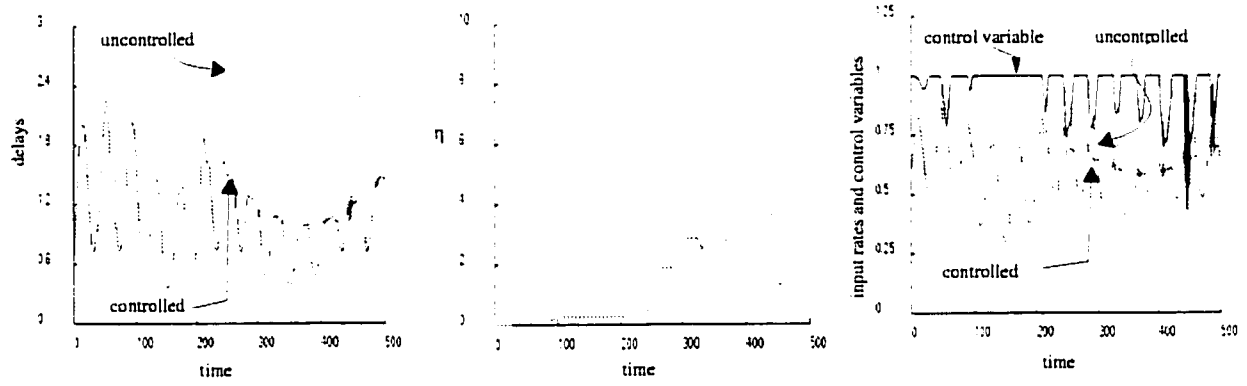



($\eta = 1, \alpha = 0.05, h = 1.3, d = 0.7$)

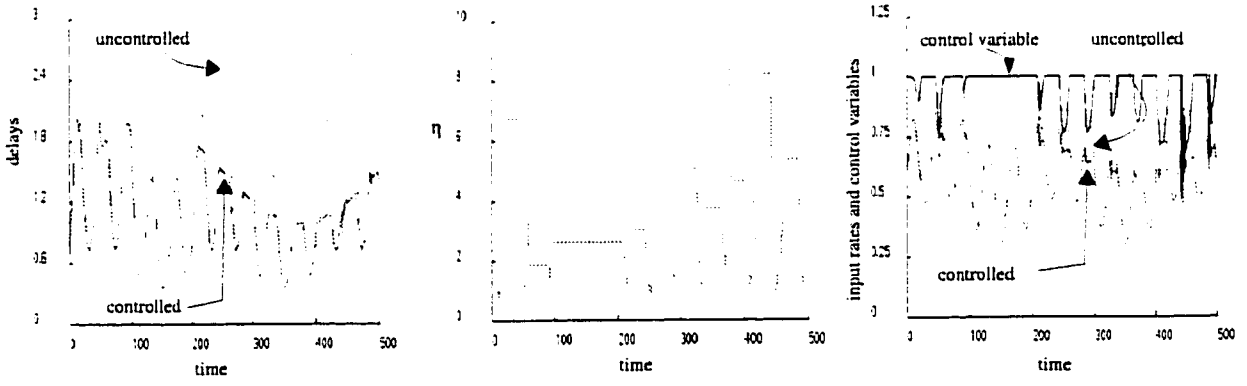
**FIGURE 3.9. Variation of an "optimally" chosen initial value of control step size during system operation.**




($\eta = 0.01, \alpha = 0.05, h = 1, d = 1$)

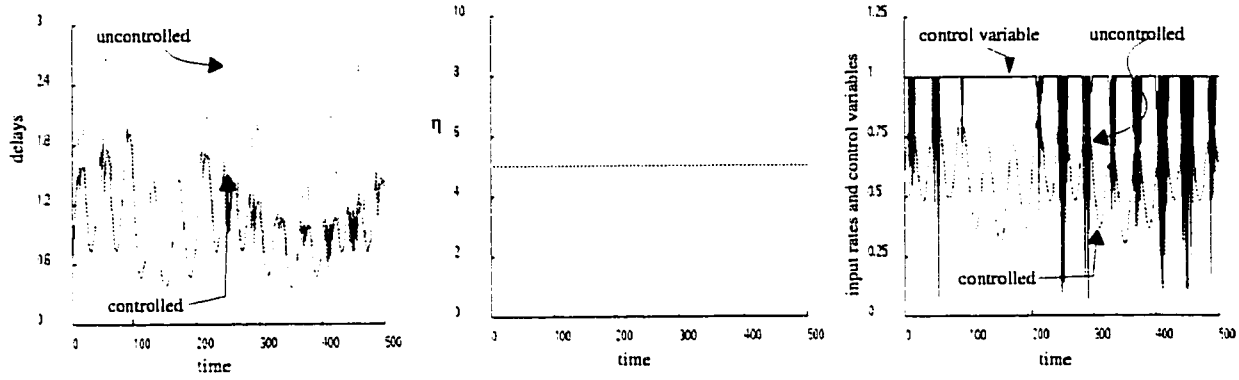



($\eta = 0.01, \alpha = 0.05, h = 1.1, d = 0.9$)

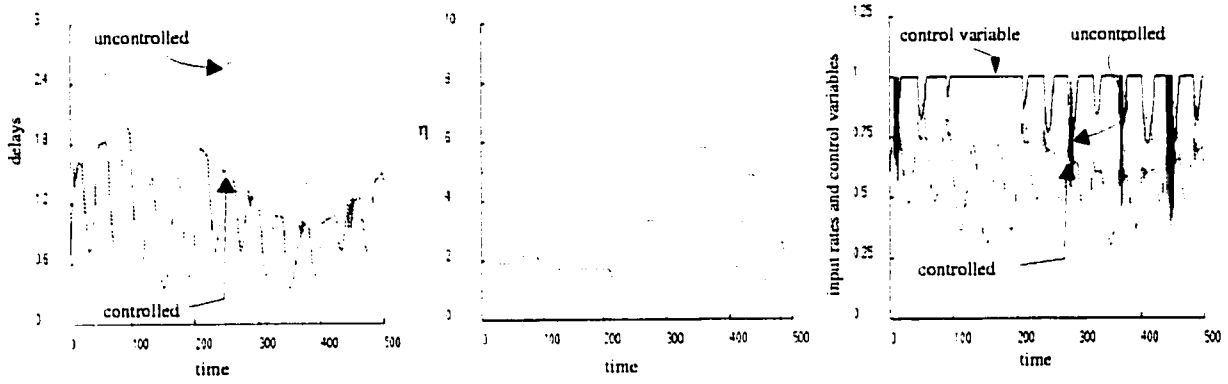



($\eta = 0.01, \alpha = 0.05, h = 1.3, d = 0.7$)

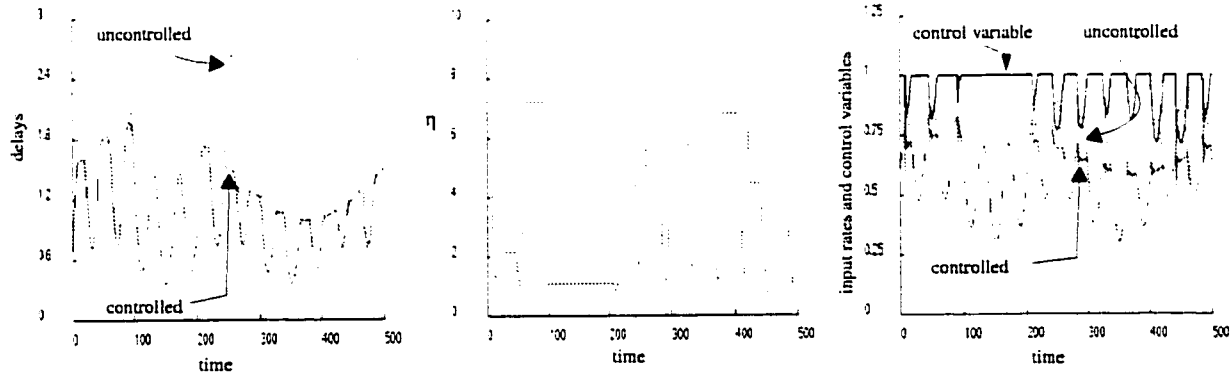
**FIGURE 3.10. Control step size adaptation applied to a case with weak control parameters.**




($\eta = 5, \alpha = 0.05, h = 1, d = 1$)




($\eta = 5, \alpha = 0.05, h = 1.1, d = 0.9$)




($\eta = 5, \alpha = 0.05, h = 1.3, d = 0.7$)

**FIGURE 3.11. Control step size adaptation applied to a case with strong control parameters.**

## 3.6. The Congestion Control Problem with Control Loop Delay

In this section, we extend the above discussion to include a congestion control framework that accounts for the effects of delayed observations in the system. The model described here reflects more closely the conditions found in computer networks where there can be significant delays in transferring information from a data source to the destination. The control models described in the next two chapters will assume the framework presented here. The objective of the discussion presented above is to test out in detail the modeling and optimization framework with a simple congestion control problem. This discussion sets the tone for the other research material to be presented in the subsequent chapters. We describe here, a congestion control framework for a low priority class of service in a computer network in which the source rates are dynamically adjusted to the amount of resources available in the network. Some data applications typically do not require a guaranteed throughput and specific end-to-end delay bounds (e.g., [ATMF96]). They can adapt to time-varying throughput (i.e., share whatever bandwidth is available) and tolerate unpredictable packet delays. For these applications, closed-loop control can be applied. We refer to the traffic generated by these applications as low-priority traffic. The nature of the low priority traffic is different from the high priority traffic (which require service with a guaranteed throughput and delay bounds). The low priority traffic will share the available bandwidth which is whatever bandwidth exists in excess of the high priority traffic (see Figure 3.12). A portion of the link bandwidth is consumed by the instantaneous high priority traffic. The low priority traffic has access to bandwidth only when no high priority traffic is waiting for transmission. Thus, the low priority traffic is allowed to use bandwidth that would otherwise be unused, increasing the link utilization without affecting the service quality of the high priority traffic.

The available bandwidth and the share of the bandwidth to each low priority traffic connection will fluctuate dynamically due to randomness in the high priority traffic and changes in the number of low priority connections sharing the available bandwidth. The low priority sources continually adapt their rates to their shares of the time-varying excess bandwidth. The performance of the network will depend on how well the sources can match their rates to the available bandwidth. Some amount of bandwidth mismatch is inevitable because delays (i.e., propagation delays, queueing delays, processing delays, etc.) are unavoidable in getting feedback information through the network. The potential mismatch between the source rates and the available bandwidth implies that buffering will be necessary in the network. Buffers will be used to absorb the low priority traffic during the momentary intervals when the source rates exceed the available bandwidth. The required amount of buffers is thus proportional to the bandwidth mismatch and feedback delay.

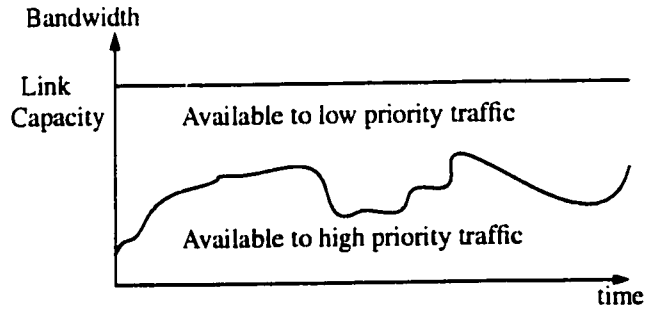



**FIGURE 3.12. Example of excess bandwidth available for low priority traffic.**

The dynamic nature of the low priority service can be seen from the feedback model shown in Figure 3.13. The buffer depicts the "bottleneck" link (the most constraining link along the connection), and the available bandwidth $\lambda(n)$ on this link can fluctuate arbitrarily between zero and the full link capacity $\lambda_{TOT}$. The source adjusts its rate $\lambda_L(n)$ to a time-delayed version of

$\lambda(n)$, that is, $\lambda_L(n) = \lambda(n-d)$, where $d$ is the round-trip delay through the network. The buffer absorbs the traffic during the periods when $\lambda_L(n) > \lambda(n)$ and releases its contents (if any) when $\lambda_L(n) \leq \lambda(n)$. It can be seen that, in the worst case, the buffer can fill up to $\lambda_{TOT} \cdot d$, which is commonly called the *bandwidth-delay product* of the network. The two factors represent the maximum possible bandwidth mismatch and the maximum duration of the mismatch, respectively. The large bandwidth-delay product in high-speed networks, especially over wide areas, implies that large queues may accumulate and hence large low priority traffic buffers will probably be necessary if packet losses must be low. Since large queues may be possible, the network cannot guarantee strict bounds on end-to-end packet delays or packet delay variations even if the low priority traffic sources adapt their rates properly. Thus, the low priority service is appropriate only for applications which can adapt their rates to the time-varying available bandwidth and tolerate unpredictable packet delays.

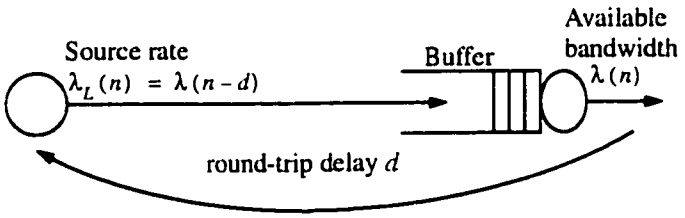
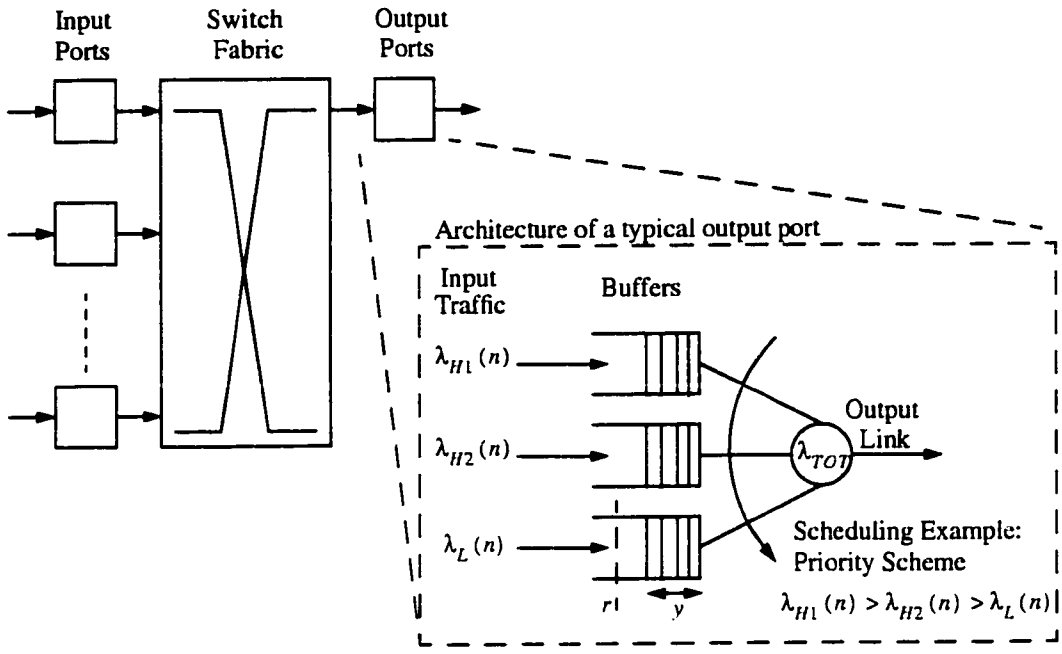



**FIGURE 3.13. Example of a source matching its rate to the available bandwidth with feedback delay.**

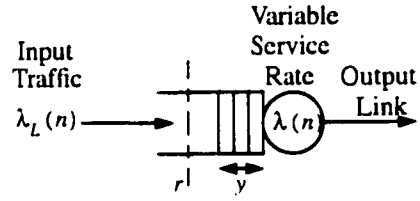
We consider a control model where a network node carries traffic from different connections having various requirements mapped into, for example, the following traffic classes: $\lambda_{H1}$, $\lambda_{H2}$, and $\lambda_L$, with $\lambda_{H1}$ as the highest class. The network node is composed of three queues dedicated to the three types of traffic. A server schedules packets according to priorities as indicated in Figure

3.14 for the $\lambda_{H1}$, $\lambda_{H2}$, and $\lambda_L$ queues. The rate control algorithm has to compute the available bandwidth left by the $\lambda_{H1}$ and $\lambda_{H2}$ traffic. The parameters and variables of the $\lambda_L$ queue for a given network node are: $y(n)$ (number of $\lambda_L$ packets waiting at time $n$), $r(n)$ (queue threshold at time $n$, usually $r(n) = r =$ constant), and $M(n)$ (number of low priority traffic connections established through the node at time $n$). The capacity of the node's output link is $\lambda_{TOT}$.

The congestion control schemes developed in the next two chapters will be based on the model described above.




a). System model (an output buffered switch example).




b). Equivalent system model for low priority traffic.

**FIGURE 3.14. A single bottleneck network node configuration.**

## 3.7. Conclusions

We have examined in this chapter how sensitivity functions using neural networks and a simple gradient descent rule can be applied to the adaptive control of congestion in queueing systems. As an alternative to the congestion control technique based on the indirect control architecture, a

direct adaptive congestion control scheme based upon a neural model and a gradient descend rule is presented. Two heuristics are described for tuning the values of the control learning rate of the gradient descend control technique to improve its performance. We described two implementations of the heuristics, namely an exponential weighted sum approach and an exponential increase/decrease method. The numerical results show that substantial improvements in system performance can be obtained when these methods are used for control. The development in this chapter and the rest of the thesis is motivated by ideas that are familiar in adaptive control. This allows us to make parallels and use insights developed in this area. The theory is based on some heuristics and is not backed up by the type of convergence and stability analysis which is commonly applied in the field of linear adaptive control. However, it becomes apparent that the neural network approach provides a conceptual basis and a procedure for nonlinear prediction and control. Some of the issues that arise in the application of adaptive control are stabilizability and solvability of the control model. Neither of these issues have been completely resolved in the linear adaptive theory and it is likely that the nonlinear problem is going to be harder to deal with. In the next two chapters, we will extend the basic congestion control model described here to account for the effects of delayed observations in the system.

# Chapter 4: Single Bit Congestion Indication and Control using Sensitivity Functions

## 4.1. Introduction

A number of binary feedback congestion control schemes have been reported in the literature [ATMF96][Bolo90][Chiu89][Rama88][Yin94]. The basic binary feedback scheme operates as follows. The network provides some information about the state of congestion by giving a binary indication of whether or not congestion has been encountered along the connection. The data source generates packets with a "congestion bit" in the packet header equal to 0 meaning no congestion. Any congested node along the path can change the congestion bit in the packet to 1 to indicate congestion at that node. The destination end system monitors the congestion bits of received packets and returns a feedback message to the source with congestion information. The source then adjusts its rate in an additive increase manner (when the feedback message indicates no congestion) or in a multiplicative decrease manner (when the feedback message indicates congestion).

This chapter presents a new binary feedback congestion control scheme in which the setting of the single bit congestion indication communicated back to the data sources by the network nodes is based upon the sign of the sensitivity of a system performance function [Awey98f][Awey98g][Awey98h]. The sign of the performance sensitivity function gives the optimal direction for the data source rate adjustment which can be done in an additive increase/multiplicative decrease manner [Chiu89][Rama88]. The proposed scheme uses a

simple neural network model of the system dynamics to determine the performance sensitivity function. In this chapter, we compare the performance of the proposed sensitivity-based binary feedback congestion control scheme to the conventional queue fill thresholding approach.

## 4.2. Congestion Bit Setting based on Queue Length

Congestion detection and congestion bit setting based on queue length is the simplest and commonly used binary feedback congestion control scheme. The congestion state is traditionally determined by the average or instantaneous number of packets queued, $q(n)$, in the internodal link or destination end system buffer at time instant $n$ (see Figure 4.1). When the queue length reaches a predetermined threshold, $r$, the packets passing through the queue will have the congestion bit set to "congestion experienced" [ATMF96][Rama88][Yin94]. The main advantage of the queue-based scheme is its low complexity, because the absolute queue length can be monitored by a single counter. However, it is not an effective method for congestion detection and control, and can produce large queues in the network nodes. The detection of congestion occurrence has to be delayed by the amount of time required to build up the queue. Similarly, the detection of congestion resolution is also delayed by the amount of time required to drain the queue. For a single queue threshold, the congestion bit $s(n)$ is 0 or 1 depending upon whether the queue at time instant $n$ is $q(n) < r$ or $q(n) \geq r$, respectively. For an additive increase/ multiplicative decrease source rate control algorithm, we have

$$\lambda(n+1) = \begin{cases} \lambda(n) + a_I, & q(n-\tau) < r \text{ or } e(n-\tau) > 0 \\ m_D \lambda(n), & q(n-\tau) \geq r \text{ or } e(n-\tau) \leq 0, \end{cases} \tag{4.1}$$

where $a_I > 0$ is an additive increase factor, $0 < m_D < 1$ is a multiplicative decrease factor, $e(n-\tau) = r - q(n-\tau)$ is an error function, and $\tau$ is the delay in conveying the queue congestion status to the source.

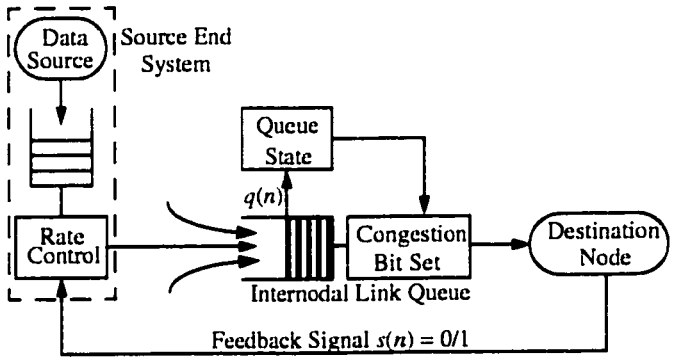



**FIGURE 4.1. Congestion bit setting based on queue length.**

## 4.3. An Optimization Approach for Congestion Bit Setting

### 4.3.1. Basic Formulation

Assume the dynamics of the queueing system can be expressed by the following nonlinear input-output equation

$$q(n+1) = f[q(n), q(n-1), \ldots, q(n-l), \lambda(n), \lambda(n-1), \ldots, \lambda(n-m)], \tag{4.2}$$

where $q(n)$ is a scalar output (i.e., queue length or queueing delay), $\lambda(n)$ is a scalar input (i.e, queue input rate), $f[.]$ is the unknown nonlinear function to be estimated by a neural network, and $l$ and $m$ are the orders of $\{q(n)\}$ and $\{\lambda(n)\}$, respectively. The purpose of an optimal control algorithm is to select a control signal $\lambda(n)$, such that the output of the system $q(n)$ is made as close as possible to a pre-specified reference $r(n)$ (usually $r(n) = r = \text{constant}$). Using the so-called series-parallel neural model [Nare90], the input to the neural network is

$$\overline{P} = [q(n), q(n-1), \ldots, q(n-l), \lambda(n), \lambda(n-1), \ldots, \lambda(n-m)], \tag{4.3}$$

and the neural model for the unknown system (4.2) can be expressed as

$$\hat{q}(n+i) = \hat{f}[q(n), \ldots, q(n-l), \lambda(n), \ldots, \lambda(n-m)], \quad i = 1, 2, \ldots, L, \tag{4.4}$$

where $\hat{q}(n+i)$ is the output of the neural network, $\hat{f}$ is the estimate of $f$, and $L$ is the prediction horizon (see Figure 4.2). If the neural network is adequately trained to learn the dynamics of the system then the squared error $(q(n+i) - \hat{q}(n+i))^2 = \varepsilon$ is small, and $\hat{q}(n+i)$ is referred to as the $i$-step ahead predicted output of the system (4.2). As a result, the control signal can be selected such that $\hat{q}(n+i)$ is made as close as possible to $r$.

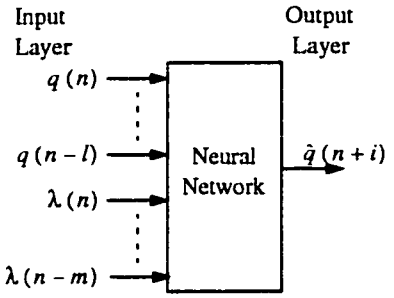



**FIGURE 4.2. Model for multi-step ahead queue state prediction.**

For our purpose, we define a cost function as follows

$$J = \frac{1}{2}e^2(n+i) = \frac{1}{2}[r-\hat{q}(n+i)]^2, \quad i = 1, 2, \ldots, L. \tag{4.5}$$

The control signal $\lambda(n)$ (i.e., the data source rate) should therefore be selected to minimize $J$. In the discrete-time case, the control variable is updated according to the following gradient descent rule.

$$\lambda(n+1) = \lambda(n) + \Delta\lambda(n) = \lambda(n) - \eta\frac{\partial J}{\partial\lambda(n)}, \tag{4.6}$$

where $\eta$ is the control step size. It can be seen that the minimization process (4.6) relies on the approximation made by the neural network to determine the appropriate control signal. Therefore, it is necessary that $\hat{q}(n+i)$ approaches the real system output $q(n+i)$ asymptotically. This can be achieved by keeping the neural network training on-line. Differentiating (4.5) with respect to $\lambda(n)$, we obtain

$$\nabla J(n) = \frac{\partial J}{\partial\lambda(n)} = -e(n+i)\frac{\partial\hat{q}(n+i)}{\partial\lambda(n)}, \quad i = 1, 2, \ldots, L, \tag{4.7}$$

where $\partial \hat{q}(n+i)/\partial \lambda(n)$ is known as the sensitivity or gradient of the system. Substituting (4.7) into (4.6), we have

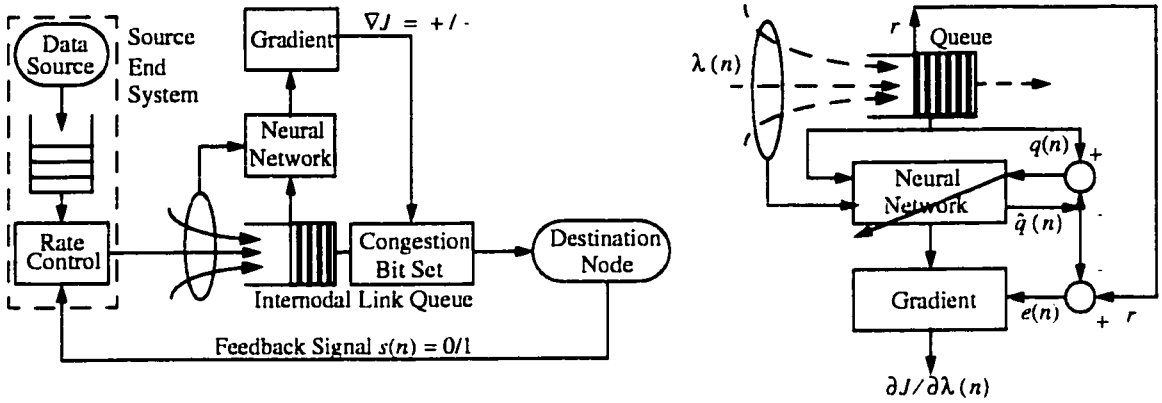
$$\lambda(n+1) = \lambda(n) + \eta e(n+i) \frac{\partial \hat{q}(n+i)}{\partial \lambda(n)}, \quad i = 1, 2, \ldots, L. \tag{4.8}$$

The gradient of the system can be analytically evaluated by using the known neural network structure as explained in Chapter 2. Rewriting (4.6) in the form

$$\lambda(n+1) = \lambda(n) - \eta \operatorname{sgn}[\nabla J(n)], \tag{4.9}$$

where $\operatorname{sgn}[\nabla J(n)]$ denotes the sign of $\nabla J(n)$ (which can be positive or negative), we can infer that (4.9) implements an additive increase/additive decrease source rate adjustment policy. So an appealing alternative to the binary congestion control scheme based on queue fill thresholding is the additive increase/ multiplicative decrease algorithm which determines the change in the data source rate $\lambda(n)$ as a function of the sign of the sensitivity of the performance index $\nabla J(n-\tau)$. That is, the congestion bit $s(n)$ is 0 or 1 depending upon whether the gradient of the system $\nabla J(n)$ at time instant $n$ is $\nabla J(n) < 0$ or $\nabla J(n) \geq 0$, respectively. The value of $\nabla J$ as computed using Eq. (4.7) gives the optimal direction for the source rate adjustment. In short, only the sign, and not the magnitude of $\nabla J$ is meaningful in this case. This sensitivity-based binary rate control scheme is illustrated in Figure 4.3. For an additive increase/multiplicative decrease source rate control algorithm, we have

$$\lambda(n+1) = \begin{cases} \lambda(n) + a_I, & \nabla J(n-\tau) < 0 \\ m_D \lambda(n), & \nabla J(n-\tau) \geq 0. \end{cases} \tag{4.10}$$




a). Closed-loop sensitivity-based congestion control scheme.

b). Sensitivity-based congestion detection scheme.

**FIGURE 4.3. An optimization approach for congestion bit setting.**

## 4.4. Numerical Results

In this section, we present numerical results to compare the performance of the queue-based and sensitivity-based congestion detection and control schemes. We consider a single connection with the following parameter set: Source peak rate $\lambda_{max}$ = 100 packets/time unit (time unit = 0.25ms), minimum rate $\lambda_{min}$ = 0 packets/time unit, additive increase factor $a_I = \lambda_{max}/16$, multiplicative decrease factor $m_D$ = 15/16. The congestion threshold $r$ for both schemes is set at $r$ = 500 packets. We consider a round trip delay of 6 time units. Two bottleneck service rate patterns are assumed in the studies: sinusoidal ($integer[35(1 + \sin(2\pi n/T)) + 10]$ packets/time unit, where $T$ is the simulation time), and random (maximum = 80 packets/time unit). Two different prediction horizons for the

bottleneck queue state are considered in this study: 1- and 3-step ahead predictions. The following neural network architecture is used: 3 layer neural network (8 inputs, 8 input neurons, 8 hidden neurons, 1 output neuron); order of $q$, $l = 3$; and order of $\lambda$, $m = 3$.

In the sensitivity-based congestion control scheme, on-line learning is necessary because the training pattern for the neural network changes with time. We propose the on-line control and neural network training algorithm given in Figure 4.4 for the sensitivity-based congestion control scheme [Awey98f][Awey98g]. The sensitivity coefficients $\partial \hat{q}(n+i)/\partial \lambda(n)$ in the algorithm can be determined using the techniques described in Chapter 2. Specifically, the algorithm described in Figure 3.5 in Chapter 3 can be used to compute the sensitivity coefficients. We define the following performance measures for the simulation run, where $T$ is the length of the simulation run:

- $q_{max} = max\{q(t): 0 \le t \le T\}$. The maximum value of $q(t)$. This indicates the buffer size required at the bottleneck to avoid packet loss.
- The time average ($\tilde{f}$) of queue size and source rate.
- Variance ($\sigma^2(f)$) of queue size and source rate.

where the time average $\tilde{f}$ and variance $\sigma^2(f)$ of a function $f(t)$ are defined by $\tilde{f} = \frac{1}{T}\int_0^T f(t)\,dt$ and $\sigma^2(f) = \frac{1}{T}\int_0^T [f(t) - \tilde{f}]^2 dt$, respectively. The simulation results are given in Figures 4.5, 4.6 and 4.7, and Table 4.1 presents a summary of the results, where

$q_{av}$ and $\lambda_{av}$ denote the time average of queue size and source rate, respectively. The following major observations can be made from the results:

- The magnitude of the queue and source rate oscillations are smaller for the sensitivity-based scheme than for the queue-based scheme. The sensitivity-based scheme with a 3-step ahead queue state prediction produces better performance than the corresponding scheme with 1-step ahead prediction. This is because for the 3-step ahead prediction case, the feedback control signals (i.e., the binary congestion bit) received at the data source from the network queue has not aged significantly and reflects more closely the network conditions when compared to the 1-step ahead prediction case. In feedback-based congestion control schemes with significant propagation delays, the control signals received at the sources can be outdated, and as a result any control response to control signals will take effect within the network only after some delay. This means the more accurately the queue state can be predicted far into the future, the more likely the congestion bit can be optimally set. A downside to the queue state prediction is that, the further into the future we want to predict, the more difficult it is to obtain predictions with small errors.

- The queue-based scheme is more sensitive to changes in the queue service rate than the sensitivity-based scheme. In the queue-based scheme, it is observed that when there is a significant decrease in the queue service rate for a length of time, the queue size grows extensively beyond the previously observed values. The evolution of the queue size is more stable for the same conditions for the sensitivity-based scheme. The variance values gives an indication of the observed quantities.

At time instant $n$:

Step 1: Get measurement of the system input and output: $\lambda(n)$ , $q(n)$

Available system history: $\lambda(n-1), \ldots, \lambda(n-m)$ ,

$q(n-1), \ldots, q(n-l)$ .

Neural network history: $\hat{q}(n), \delta_i^{(p)}(n-1), x_i^{(p)}(n-1)$ ,

for all layers $p$ and all nodes $i$.

Step 2: Neural network feedforward to compute $\hat{q}(n+i)$ .

Neural network backpropagation to compute $\partial\hat{q}(n+i)/\partial\lambda(n)$ .

Step 3: Compute gradient function $\nabla J(n) = \partial J/\partial\lambda(n)$ .

Set congestion bit of data packet = $\begin{cases} 1, & \nabla J(n) \geq 0 \\ 0, & \text{otherwise.} \end{cases}$

Step 4: Using $[\hat{q}(n) - q(n)]$ as neural network error to update neural network weights:

$$\Delta w_{ij}^{(p)}(n) = -\beta[\hat{q}(n) - q(n)]\delta_i^{(p)}(n-1)x_j^{(p-1)}(n-1) + \gamma\Delta w_{ij}^{(p)}(n-1) ,$$

$$w_{ij}^{(p)} \Leftarrow w_{ij}^{(p)} + \Delta w_{ij}^{(p)}(n) ,$$

for $p = p_f, p_{f-1}, \ldots, 1$ , $i = 1, 2, \ldots, m_{(p)}$ , $j = 1, 2, \ldots, m_{(p-1)}$ .

$\beta$ - Neural network learning rate; $\gamma$ - Neural network momentum constant.

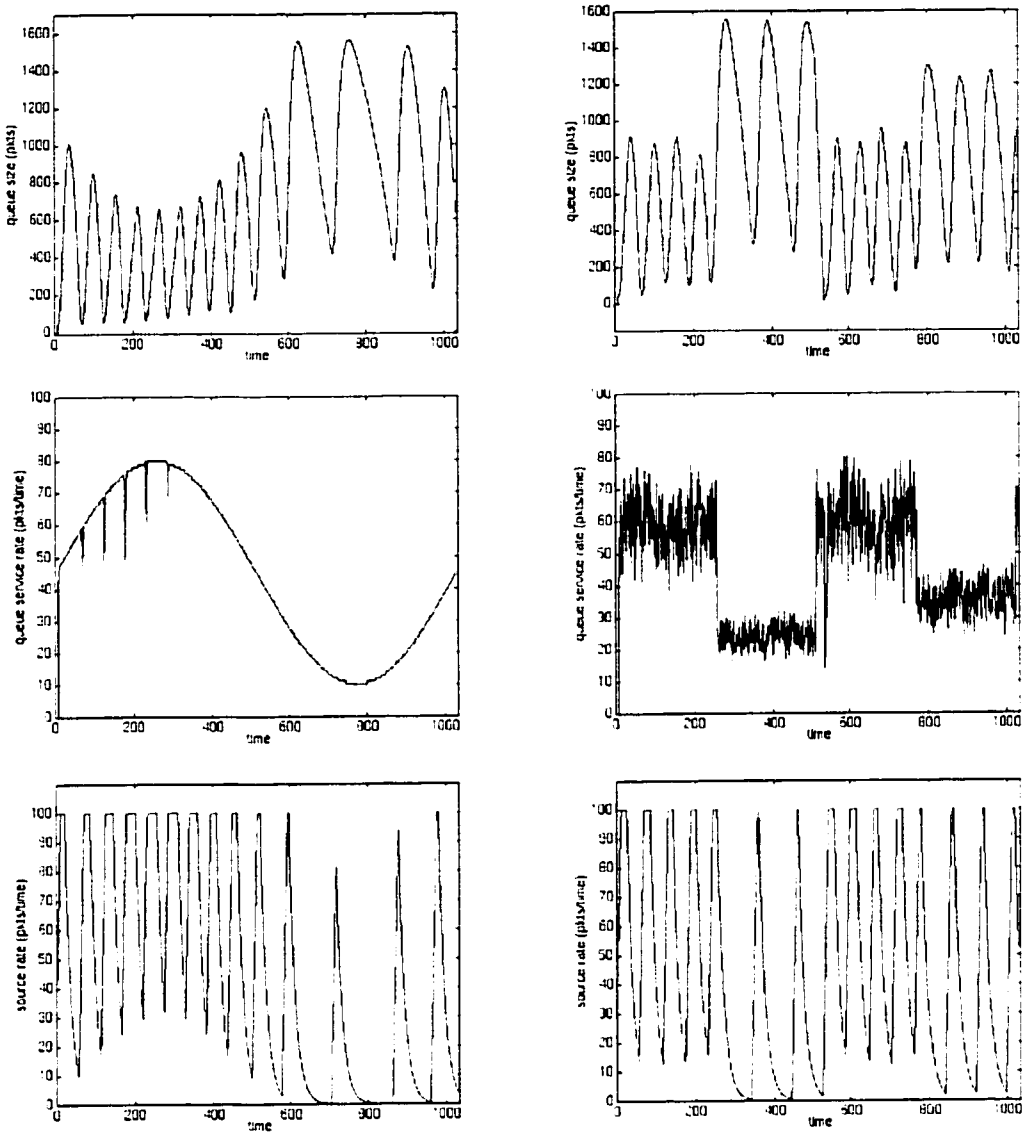
Step 5: Next interval, $n \Leftarrow n+1$ , go to Step 1.

Note: Steps 3 and 4 can be done in parallel.

**FIGURE 4.4. Algorithm for congestion bit setting and neural network training.**

**TABLE 4.1. Summary of simulation results.**

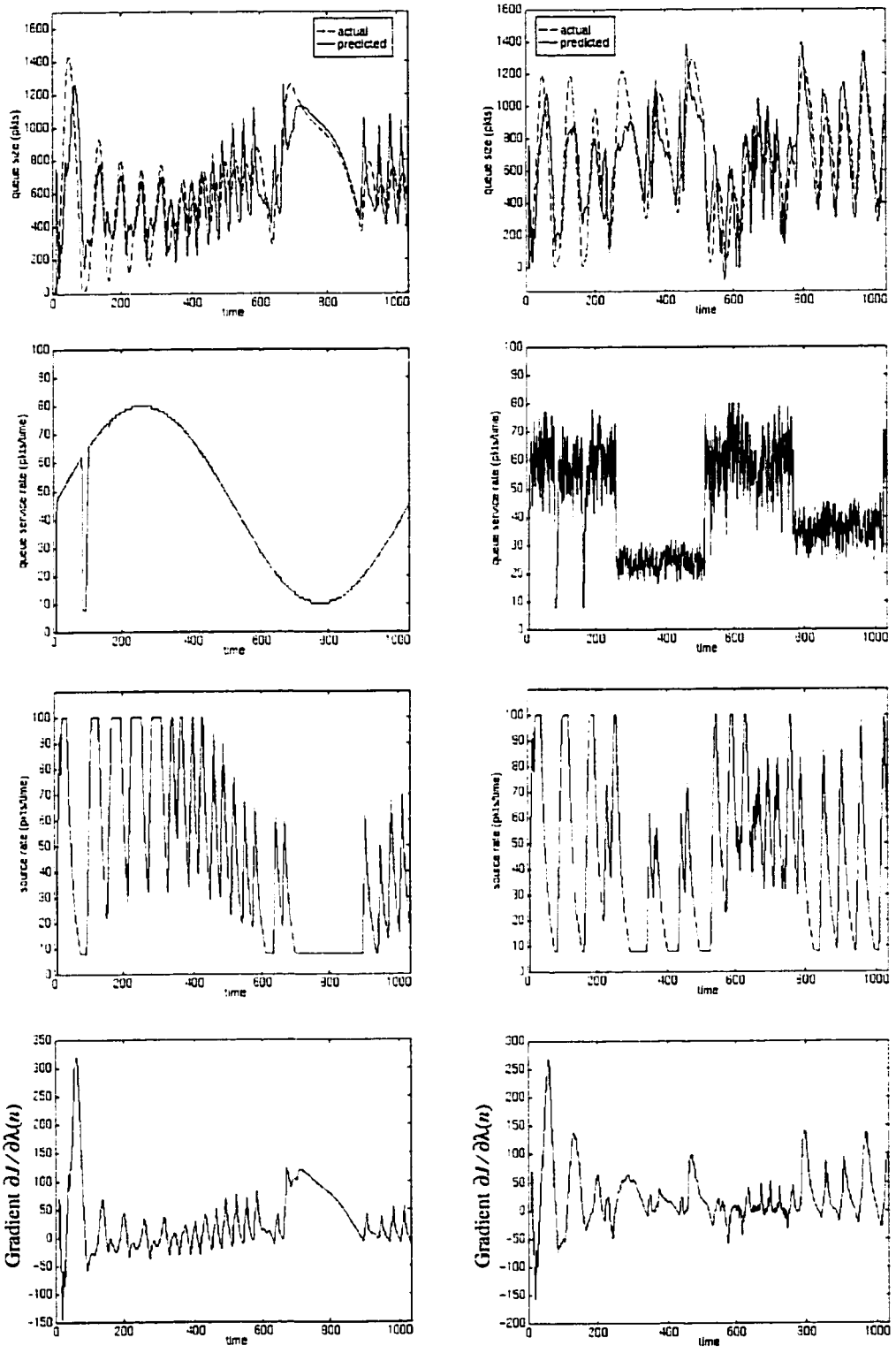
| Approach | Service Type | $q_{max}$ | $q_{av}$ | $\sigma^2(q)$ | Time index of $q_{max}$ | $\lambda_{av}$ | $\sigma^2(\lambda)$ |
|---|---|---|---|---|---|---|---|
| Queue-based | Random | 1560 | 724.5 | 187569.3 | 757 | 45.1 | 1370.1 |
| | Sine | 1556 | 713.8 | 179305.5 | 281 | 45.7 | 1211.4 |
| Gradient: 1-Step | Random | 1293 | 666.2 | 108986.4 | 480, 481, 482 | 44.4 | 878.3 |
| | Sine | 1428 | 653.3 | 86115.5 | 48 | 44.4 | 1066.9 |
| Gradient: 3-Step | Random | 1307 | 622.9 | 88526.7 | 800 | 44.2 | 929.9 |
| | Sine | 1443 | 628.7 | 89936.5 | 49 | 44.5 | 1139.7 |




a). Bottleneck queue with a "sinusoidal" service rate.

b). Bottleneck queue with a "random" service rate.

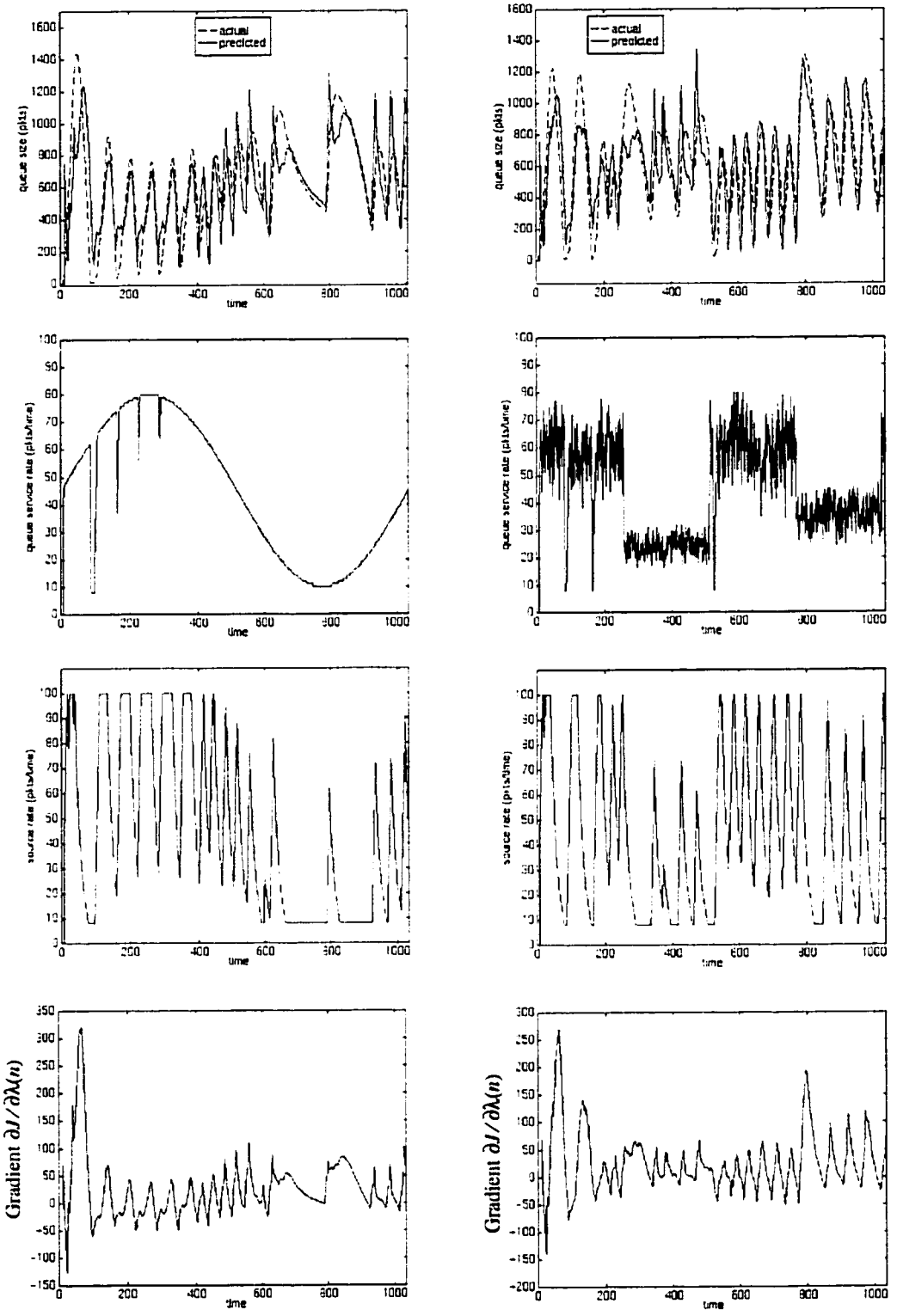
**FIGURE 4.5. Performance of the queue-based scheme for congestion bit setting.**




a). Bottleneck queue with a "sinusoidal" service rate.

b). Bottleneck queue with a "random" service rate.

**FIGURE 4.6. Sensitivity-based scheme with 1-step ahead queue state prediction.**




a). Bottleneck queue with a "sinusoidal" service rate.

b). Bottleneck queue with a "random" service rate.

**FIGURE 4.7. Sensitivity-based scheme with 3-step ahead queue state prediction.**

## 4.5. Conclusions

We have described in this chapter a new neural network-based technique for optimal congestion bit setting in a binary feedback flow control scheme for computer networks. This technique employs the sensitivity of the system performance to generate feedback from the network to the data sources. The optimal direction for rate adjustment at the source is based on a single bit feedback signal which depends upon the sign of the sensitivity of the system performance index with respect to queue input rate. We have presented simulation results on the performance of the proposed scheme and the results show that the sensitivity-based scheme has better performance compared to the conventional queue thresholding scheme. The sensitivity-based scheme has smaller oscillations in queue size and source rates and the oscillations do not vary significantly even if there are large changes in bottleneck queue service rates.

The sensitivity-based approach for congestion detection and bit setting is not intended to solve the limitations of single-bit binary congestion control. A problem with single-bit congestion detection and control schemes is that they generally result in large oscillations in queue size. The main reason is that the single congestion bit can indicate only the existence of congestion somewhere along the data connection, but not the location or severity. It also takes a complete round-trip before the congestion information returns to the source end-system. Most implementations of the basic binary congestion control scheme have been for sources which share a common buffer at the network node and the service discipline at the node is First-Come-First-Served (FCFS). An important reason for considering the FCFS implementation is its simplicity and low cost. However, a congested switch has

no ability to mark sources selectively. In particular, a switch will mark all sources regardless of the individual source's rates. This gives rise to the "beat down" problem, where a connection traversing multiple hops is subject to being marked more often than those traversing fewer congested links (owing to the unsynchronized congestion waves at each of these switches), with the result that its bandwidth is driven down for unacceptably long periods of time [Benn94]. This undesirable effect is proportional to the number of traversed congested links. The limitation of non-selective binary marking mechanisms implies the need for some form of intelligent marking to achieve fair distribution of available bandwidth [RobeL94][SiuK96]. The above issues have been extensively examined in the literature (e.g., [Arul96]) and so will not be considered here.

The studies here show that the sensitivity-based approach is able to reduce the magnitude of the queue oscillations (as indicated by the variance values in Table 4.1) but cannot completely eliminate the oscillations. Using the sensitivity of the system performance function makes it possible to detect congestion early and this leads to early feedback control reaction by the data sources. However, the sensitivity-based approach is dependent on finding neural architectures and training schemes suitable for modeling the dynamics of the network queues. The neural architecture used in the studies is a simple one and it is believed that even better performance can be obtained if good architectures capable of modeling the system dynamics can be obtained.

# Chapter 5: Multi-Step Predictive Techniques for Congestion Control

## 5.1. Introduction

Feedback-based congestion control in high-speed computer networks sometimes suffer from large propagation delays in data transfers which implies that the control information received at the data sources or the network access points from the network nodes are outdated, and that any response to control commands will take effect within the network only after some time delay. Thus, the design of an effective congestion control mechanism is difficult. In this chapter, we address the issue of control loop delays through an adaptive predictive technique using neural networks.

The control approach adopted in this chapter is as follows: The level of network congestion is monitored through the occupancy or queueing delay $y$ of a network buffer, with the control target being some threshold $r$. Based on the difference between $y$ and $r$, the congestion controller periodically calculates an admission rate and supplies the sources with the result of the calculation. In turn, the traffic sources reduce their transmission rate to the level allowed by the controller. The choice of the setpoint $r$ reflects a trade-off between mean packet delay, packet loss and bandwidth loss (which is the bandwidth a source loses because it has no data to send when it is eligible for service).

Our main contribution in this chapter is the development of a multi-step adaptive predictive technique for the control of the state of a network queue in which the prediction is

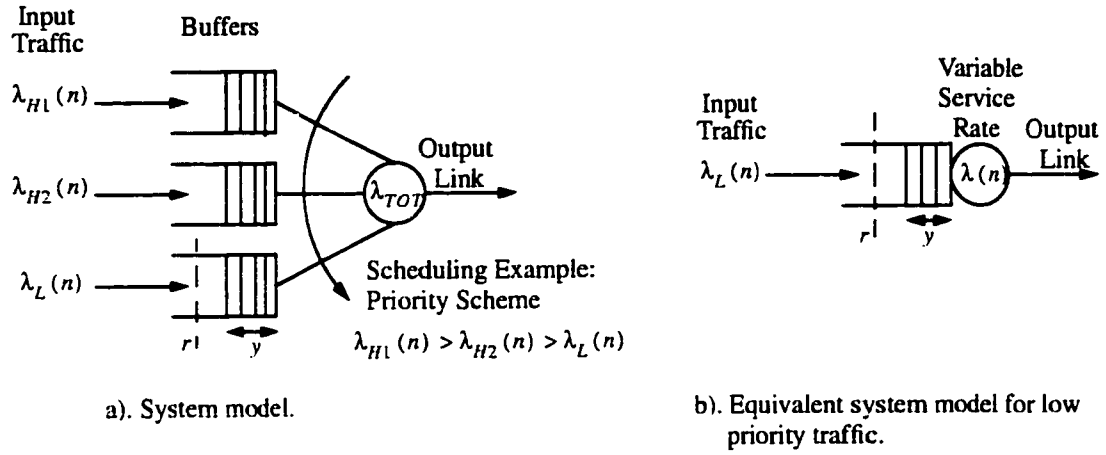
made using neural networks [Awey98i]. We demonstrate through simulations that, the technique provides good control adaptability and performance. Most of the neural network-based congestion control schemes proposed in the literature do not consider the effect of control loop delays in the development of the control algorithms because the models become too complex and intractable. In spite of the complexity introduced in the analysis by the control loop delay, the aim of this chapter is to investigate the feasibility of using neural networks and predictive techniques to solve the feedback-based congestion control problems.

## 5.2. Congestion Control Model

In this section, we describe a congestion control framework for a low priority class of service in a computer network in which the source rates are dynamically adjusted to the amount of resources available in the network. The discussion here is based on the congestion control framework described in Section 3.6 of Chapter 3. The low priority traffic will share the available bandwidth, which is whatever bandwidth exists in excess of the high priority traffic. A portion of the link bandwidth is consumed by the instantaneous high priority traffic. The low priority traffic has access to bandwidth only when no high priority traffic is waiting for transmission. Thus, the low priority traffic is allowed to use bandwidth that would otherwise be unused, increasing the link utilization without affecting the service quality of the high priority traffic. The low priority sources must continually adapt their rates to their shares of the time-varying excess bandwidth.

The congestion control scheme described in this chapter provides a means for the dynamic evaluation of the level of available resources left unused in the network in the presence of high priority traffic. The evaluation and distribution (or resource sharing) functions for computing the rate allocation to each individual source are based on a neural network-based control protocol that controls the filling level of the low priority traffic buffer. The network node buffer allocated to the low priority service is used in order to obtain a better statistical gain in the network.

We consider a control model where a network node carries traffic from different connections having various requirements mapped into, for example, the following classes: $\lambda_{H1}$, $\lambda_{H2}$, and $\lambda_L$, with $\lambda_{H1}$ as the highest class and $\lambda_L$ as the lowest. For this example, the network node is composed of three queues dedicated to the three types of traffic. A server schedules packets according to priorities as indicated in Figure 5.1 for the $\lambda_{H1}$, $\lambda_{H2}$, and $\lambda_L$ queues. The rate control algorithm has to compute the available bandwidth left by the $\lambda_{H1}$ and $\lambda_{H2}$ traffic (i.e., the high priority traffic). The parameters and variables of the $\lambda_L$ queue for a given network node used for congestion control are: $y(n)$ (number of $\lambda_L$ packets waiting at time $n$), $r(n)$ (queue threshold at time $n$, usually $r(n) = r =$ constant), and $M(n)$ (number of low priority traffic connections established through the node at time $n$). The capacity of the node's output link is $\lambda_{TOT}$.




a). System model.

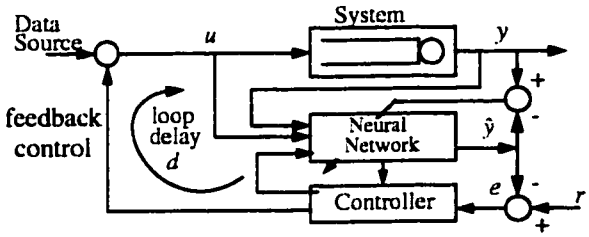
b). Equivalent system model for low priority traffic.

**FIGURE 5.1. Basic congestion control model for a single bottleneck network node configuration.**

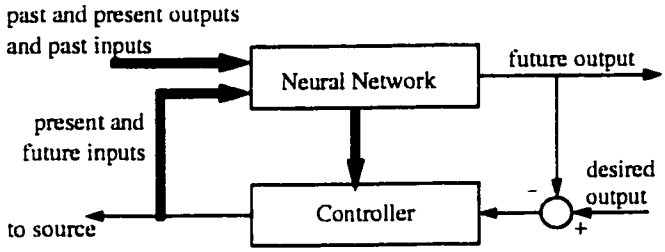
Figure 5.2 presents the control structure for the congestion control problem considered here. To help cast the control problem into a form suitable for the application of adaptive control methods, we consider initially a single source transmitting data to a network buffer and a fixed control loop delay *d*.

The congestion control can be broken into three processes:

- measuring congestion within the network (using neural network prediction),
- exchanging information between nodes (using control packets), and
- restricting the flow of packets into the network through source rate adaptation based on a set of admissible controls computed by the rate algorithm.




a). Closed-loop congestion control scheme.




b). Predictive control model.

**FIGURE 5.2. Congestion control scheme for a system with control loop delay.**

Predictive controllers generally have four major features [Soet92]:

- A *model* of the process to be controlled. In our control problem, this model is used to predict the process output over a prediction horizon.
- The *criterion* or *cost function* that is minimized in order to obtain the optimal controller output sequence over the prediction horizon. Usually, a quadratic criterion which weights tracking error and sometimes controller output is used.
- The *reference trajectory* for the process output.
- The *minimization procedure* itself.

Various controllers with distinctive characteristics may be designed by appropriate choice of these features. Let $u(n)$ denote the control variable (i.e., rate at which the source must transfer data). The performance index is usually of the general form [Sage68]

$$J = \int_{t_0}^{t_f} C\Big( \underline{x}(t), \underline{x}^e(t), u(t) \Big) dt, \tag{5.1}$$

where $\underline{x}(t)$ is the vector of state variables of the controlled process and $\underline{x}^e(t)$ the desired state. $C$ denotes the cost function, and it can be defined as the distance between the actual system state $\underline{x}(t)$ and the desired state $\underline{x}^e(t)$ . $C$ is minimized by selecting the optimal control $u^*(t)$, $t_0 \le t \le t_f$. Since the controlled process is stochastic, $C$ should be considered as an expected value. If we consider directly the minimization of $C$, a large amount of computation is involved for each iteration. The common way to reduce the amount of computation per iteration is to base the control variable updates on the instantaneous performance index

$$J = C\Big( \underline{x}(t), \underline{x}^e(t), u(t) \Big). \tag{5.2}$$

A well-known example of a minimization procedure is the gradient descent algorithm. In the discrete-time case, the control variable is updated according to the rule

$$u(n+1) = u(n) - \eta \frac{\partial J}{\partial u(n)}, \tag{5.3}$$

where $\eta > 0$ is the *control step size*. The sensitivity of the performance index $\nabla J(n) = \partial J / \partial u(n)$ is analytically evaluated using a known neural network structure as described in Chapter 2.

## 5.3. Computation of the Total Available Bandwidth for the Low Priority Traffic

Let us denote $u(n)$ as the total available bandwidth at time $n$ left to the low priority traffic $\lambda_L$. The available bandwidth $u(n)$ is a function of the node utilization as well as the $\lambda_{H1}$ and $\lambda_{H2}$ traffic. We expect that the available rate $u(n)$ will be equal to the bandwidth not used by other traffic with higher priority plus the bandwidth needed to fill up the buffer capacity up to the setpoint $r$ at a given instant $n$. Assume the nonlinear network system can be described by the following discrete-time equation

$$y(n) = f[y(n-1), \ldots, y(n-l), u(n-d), \ldots, u(n-d-m)], \tag{5.4}$$

where $y(n)$ is a scalar output (i.e., queueing delay, queue length, etc.), $u(n)$ is a scalar input, $f[.]$ is the unknown nonlinear function to be estimated by a neural network, $l$ and $m$ are the orders of $\{y(n)\}$ and $\{u(n)\}$, respectively, and $d \geq 1$ is the control loop delay. The purpose of the control algorithm is to select a control signal $u(n-d)$, such that the output of the system $y(n)$ is made as close as possible to a pre-specified reference $r(n)$. The neural model for the unknown system (5.4) can be expressed as

$$\hat{y}(n+1) = \hat{f}[y(n), \ldots, y(n-l+1), u(n-d+1), \ldots, u(n-d-m+1)], \tag{5.5}$$

where $\hat{y}(n+1)$ is the one-step ahead predicted output of the neural network and $\hat{f}$ is the estimate of $f$. If the neural network is adequately trained to learn the dynamics of the system then the squared error $(y(n+1) - \hat{y}(n+1))^2 = \varepsilon$ is small, and $\hat{y}(n+1)$ is referred to as the predicted output of the system (5.4). As a result, the control signal $u(n-d+1)$ can be selected such that $\hat{y}(n+1)$ is made as close as possible to $r(n+1)$.

Since the neural model represents the system to be controlled asymptotically, it can be used to predict future values of the system output. For this purpose, let us denote $L \geq d$ as the *prediction horizon* and

$$R = [r(n+d), r(n+d+1), \ldots, r(n+L)]^T \tag{5.6}$$

as the future values of the set-point, with $[.]^T$ denoting the transpose of the vector (matrix) [.], and

$$\hat{Y} = [\hat{y}(n+d), \hat{y}(n+d+1), \ldots, \hat{y}(n+L)]^T \tag{5.7}$$

as the predicted output of the system using the neural network model, then the following error vector

$$E = [e(n+d), e(n+d+1), \ldots, e(n+L)]^T \tag{5.8}$$

can be obtained where

$$e(n+i) = r(n+i) - \hat{y}(n+i), \; d \leq i \leq L. \tag{5.9}$$

Define the control signals to be determined as

$$U = [u(n), u(n+1), \ldots, u(n+N-1)]^T, \tag{5.10}$$

where $N$ is the *control horizon* (which we choose in this thesis to be $N = L - d + 1$ in order to obtain square matrices in our control equations), and assume the following objective function

$$J = \frac{1}{2}\sum_{i=d}^{L} [r(n+i) - \hat{y}(n+i)]^2 = \frac{1}{2}[E^T E]. \tag{5.11}$$

Then our objective is to find $U$ such that $J$ is minimized. Using the gradient descent rule, we obtain the following expression

$$U^{k+1} = U^k - \eta \frac{\partial J}{\partial U^k}, \tag{5.12}$$

$$\frac{\partial J}{\partial U^k} = -\frac{\partial \hat{Y}^k}{\partial U^k} E^k, \tag{5.13}$$

where the matrix of the dynamic sensitivity derivatives $\partial \hat{y}(n+i)/\partial u(n+j)$, $d \le i \le L$, $0 \le j \le N-1$ is given by

$$\frac{\partial \hat{Y}^k}{\partial U^k} = \begin{bmatrix} \frac{\partial \hat{y}^k(n+d)}{\partial u^k(n)} & \frac{\partial \hat{y}^k(n+d+1)}{\partial u^k(n)} & \cdots & \frac{\partial \hat{y}^k(n+L)}{\partial u^k(n)} \\ 0 & \frac{\partial \hat{y}^k(n+d+1)}{\partial u^k(n+1)} & \cdots & \frac{\partial \hat{y}^k(n+L)}{\partial u^k(n+1)} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \frac{\partial \hat{y}^k(n+L)}{\partial u^k(n+N-1)} \end{bmatrix}, \tag{5.14}$$

$$U^k = \left[u^k(n), u^k(n+1), \ldots, u^k(n+N-1)\right]^T,$$

and

$$E^k = \left[e^k(n+d), e^k(n+d+1), \ldots, e^k(n+L)\right]^T.$$

In the above equations, $k$ is the iteration index ($k = 1, 2, \ldots$). The control sequence $u^{k-1}(n+i)$, $i = 0, \ldots, N-1$, is used to generate the prediction sequence $\hat{y}^k(n+j)$, $j = d, \ldots, L$. For $k = 1$, $u^0(n+i)$ is initialized by using the control sequence generated in the last sampling period. Then, the performance index $J$ is minimized iteratively until the control sequence which leads to the extremum of $J$ is found. The gradient descent algorithm can be expressed as

$$u^{k+1}(n+i) = u^k(n+i) + \eta I_i \frac{\partial \hat{Y}^k}{\partial U^k} E^k, \tag{5.15}$$

$u^0(n+i) = u(n-1+i)$, $i = 0, 1, \ldots, N-1$, $k = 1, 2, \ldots$, where $u^k(n+i)$ denotes the $k$th iteration value of the control $u(n+i)$ and $I_i$ is a row vector with a 1 in the $i$th ($i = 1, 2, \ldots, N$) column and all zeros elsewhere.

According to the principle of receding horizon control [Soet92], only the first control signal

$$u(n+1) = u(n) + \eta \sum_{i=d}^{L} e(n+i) \frac{\partial \hat{y}(n+i)}{\partial u(n)} \tag{5.16}$$

of the determined control sequence $U$ is applied to the process. In each sampling period, one iteration is carried out. The performance index can also be formulated in terms of Clarke's weighted predictive control performance index [ClarW87] as described in Appendix D. The receding horizon control approach can be summarized by the following steps:

1. Predict the system output $\hat{Y}$ over the range of future times (i.e., the prediction horizon).
2. Assume that the future desired outputs $R$ are known. In the congestion control problem, $R$ is known in advance and is normally a fixed threshold, but the technique described here allows $R$ to be dynamic if necessary.
3. Choose a set of future controls $U$ which minimize the future errors between the predicted future output and the future desired output. The minimization is done using the gradient descent rule.
4. Use the first element of $U$ as the next control input and repeat the whole process at the next instant.

## 5.4. Prediction Models

In order to predict the process output over the prediction horizon, a multi-step ahead predictor is required. Clearly, multi-step ahead predictors depend heavily on the model of the process. Some of the models that are often used in predictive controllers are the Finite Impulse Response (FIR) model, the Finite Step Response (FSR) model, and the Transfer Function model [Soet92]. These models typically require many parameters of the process to be known or estimated which makes them not suitable for modeling processes in high-

speed networks. In this section we show how multi-step ahead predictors can be obtained using neural network models of the process.

Let us assume that the neural model used to construct the predictors is described by the following equation

$$\hat{y}(n) = g\left[W^{(p)} g\left[W^{(p-1)} \ldots g\left[W^{(1)} V(n-1)\right]\right]\right], \tag{5.17}$$

where $W^{(j)}$ is the synaptic weight vector in layer $j = 1, 2, \ldots, p$, $g(.)$ is the activation function of a neuron, and

$$V(n-1) = \{y(n-1), \ldots, y(n-l), u(n-d), \ldots u(n-d-m)\}.$$

For each node in the feedforward neural network, we have

$$z_i^{(p)} = \left(W_i^{(p)}\right)^T X^{(p-1)}, \tag{5.18}$$

$$x_i^{(p)} = g\left(z_i^{(p)}\right) = g\left[\left(W_i^{(p)}\right)^T X^{(p-1)}\right], \tag{5.19}$$

where $z_i^{(p)}$ and $x_i^{(p)}$ are respectively the summed input and output of the $i$th node in the $p$th layer, $W_i^{(p)} = \left[w_{i,1}^{(p)}, \ldots, w_{i,m_{(p-1)}}^{(p)}\right]^T$ is the vector of synaptic weights, $X^{(p-1)} = \left[x_1^{(p-1)}, \ldots, x_{m_{(p-1)}}^{(p-1)}\right]^T$ is the vector of all the outputs in the previous layer, $m_{(p-1)}$ is the number of inputs to the $i$th node in the $p$th layer, and $g(.)$ is a single-valued monotonic nonlinear sigmoid function, e.g.,

$$g(x) = \tanh(x) = \frac{1 - e^{-x}}{1 + e^{-x}}. \tag{5.20}$$

The calculation of the gradient of the performance index, $\partial J / \partial U$, requires the prediction of the system output over the range of future times (i.e., the prediction horizon). Below we consider two neural network architectures for predicting the system output: one with all the predictions made at the same time at different nodes of the output layer of a single neural network, and another which employs a single neural network with one output and using a recursive one-step ahead prediction. In the recursive case, in order to have predictions for the different sampling intervals in the future, the neural network needs to be iterated, i.e., the neural network needs to be linked back to itself to go as far as needed in the future.

### 5.4.1. Recursive Multi-step Ahead Predictor

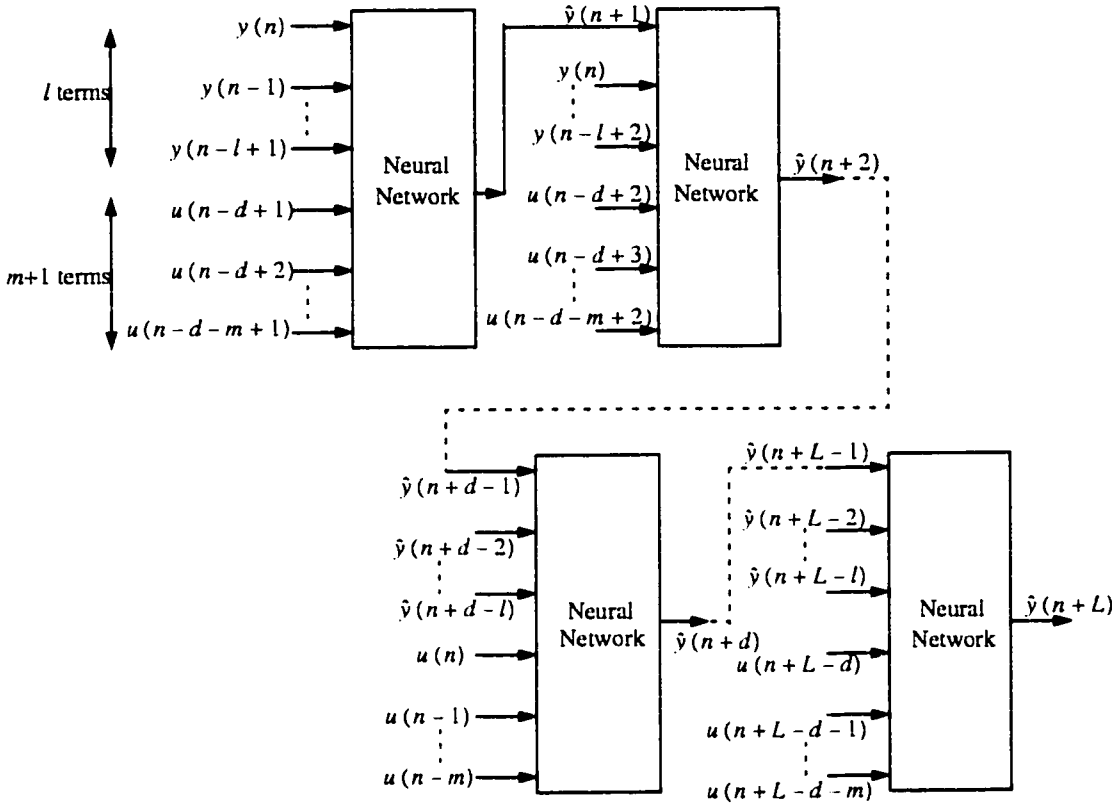



**FIGURE 5.3. Recursive $L$-step ahead prediction.**

Let us assume that the data transfer process can be described by the following nonlinear model

$$\hat{y}(n+1) = f[y(n), \ldots y(n-l+1), u(n-d+1), \ldots, u(n-d-m+1)] . \quad (5.21)$$

Based on this model, we can use a recursive technique to obtain the recursive $d$-step ahead predictor, i.e.,

$$\hat{y}(n+d) = \hat{f}[\hat{y}(n+d-1), \ldots \hat{y}(n+d-l), u(n), \ldots, u(n-m)] . \quad (5.22)$$

Clearly, this predictor depends on the predictions at the previous steps within the prediction horizon. Suppose the maximum horizon of the prediction is $L$, the $i$-step-ahead recursive predictor can be described as follows

$$\hat{y}(n+i) = \hat{f}[\hat{y}(n+i-1), \ldots, \hat{y}(n+i-l), u(n+i-d), \ldots, u(n+i-d-m)], \; 1 \le i \le L \tag{5.23}$$

A neural network representation of the above nonlinear predictive method is given as follows

$$\hat{y}(n+i) = g\left[W^{(p)} g\left[W^{(p-1)} \ldots g\left[W^{(1)} V_r(n+i-1)\right]\right]\right], \quad 1 \le i \le L, \tag{5.24}$$

where

$$V_r(n+i-1) = \{\hat{y}(n+i-1), \ldots, \hat{y}(n+i-l), u(n+i-d), \ldots u(n+i-d-m)\}$$

is the input vector. If $i-k \le 0$, $k = 1, 2, \ldots, l$, then $\hat{y}(n+i-k) = y(n+i-k)$. The $L$-step ahead prediction is illustrated in Figure 5.3. In general, the control sequence $u^{k-1}(n+j)$, $j = 0, 1, \ldots, N-1$, is used to generate the prediction sequence $\hat{y}^k(n+j)$, $j = d, \ldots, L$.

Suppose the feedforward neural network used to predict the $d$-step ahead system output $\hat{y}(n+d)$ is shown in Figure 5.4. Let us define the following parameters:

Total number of layers, $p_t$;

Layers are numbered, $p = 1, 2, \ldots, p_t$;

Neurons are numbered, $i = 1, 2, \ldots, m_{(p)}$;

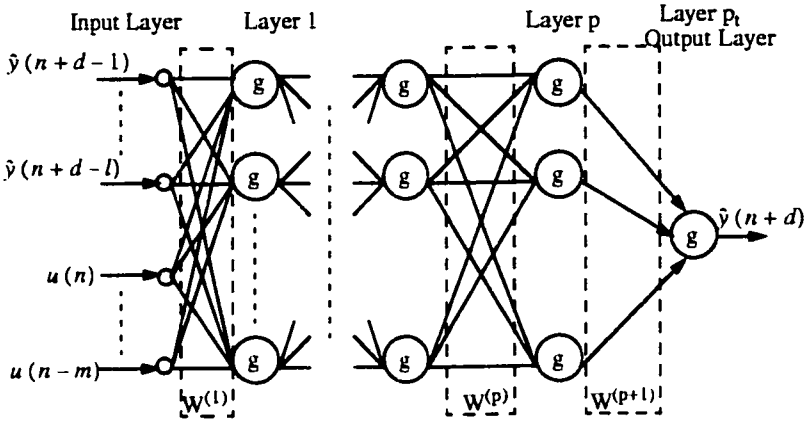
Input layer, $\bar{P} = X^{(0)}$, where

$$\bar{P}(n) = [\hat{y}(n+d-1), \ldots, \hat{y}(n+d-l), u(n), \ldots, u(n-m)]^T;$$

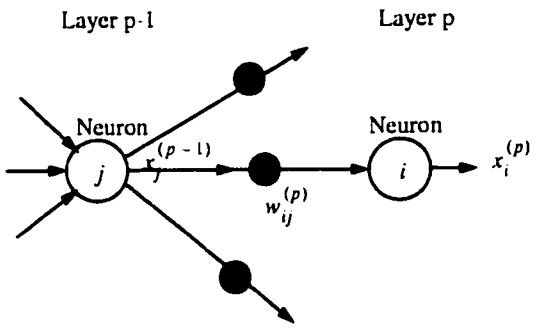
Output layer, $\hat{y}(n+d) = X^{(p_t)}$.

For each neuron $x_i^{(p)} = g\left(z_i^{(p)}\right)$, where $z_i^{(p)} = \sum_{j=1}^{m_{(p-1)}} w_{ij}^{(p)} x_j^{(p-1)}$.

A key requirement of the neural predictive controller is the computation of the dynamic sensitivity derivatives $\partial\hat{y}(n+i)/\partial u(n+k)$, $d \le i \le L$, $0 \le k \le N-1$. The dynamic sensitivity derivative $\partial\hat{y}(n+d)/\partial u(n)$, for instance, can be computed using the algorithm in Figure 5.5. This algorithm is derived based on the discussions on sensitivity analysis in Chapters 2 and 3.




a). Feedforward neural model for deriving $\partial\hat{y}(n+d)/\partial u(n)$.




b). Interconnection of neurons.

**FIGURE 5.4. Neural model for computing the dynamic sensitivity derivatives at $d$ time steps ahead.**

Algorithm for computing $\partial \hat{y}(n+d)/\partial u(n)$ :

Define local gradient for layer $p$, neuron $i$:

$$\delta_i^{(p)} = \begin{cases} g\left(z_i^{(p)}\right), & \text{if } p = p_t \\ \sum_{j=1}^{m_{(p+1)}} \left(\delta_j^{(p+1)} w_{ji}^{(p+1)}\right) g\left(z_i^{(p)}\right), & \text{if } 1 \le p < p_t, \end{cases}$$

where the input layer $X^{(0)} = P(n)$ .

Computation Steps:

Step 1: Compute $\delta_j^{(p)}$ , $j = 1, 2, \ldots, m_{(p)}$ , starting from $p = p_t$ , then backpropagate in neural network recursively to obtain $\delta_j^{(p)}$ for $p = p_t - 1, p_t - 2, \ldots, 2, 1$ .

Step 2: Suppose $u(n) = x_i^{(0)}$ (i.e., element $i = l + 1$ of $X^{(0)}$ ), then

$$\frac{\partial \hat{y}(n+d)}{\partial u(n)} = \sum_{j=1}^{m_{(1)}} \delta_j^{(1)} w_{ji}^{(1)} .$$

**FIGURE 5.5. Algorithm for computing the dynamic sensitivity derivatives at *d* time steps ahead.**

From Figure 5.3, we can see that the calculation of the dynamic sensitivity derivatives $\partial \hat{y}(n+i)/\partial u(n+k)$ , $d \le i \le L$, $0 \le k \le N-1$ is more complex. The influence of a change in the control input $u$ on any future output $y$ is to be determined. Given the fact that the neural network has both $y$ and $u$ inputs, this influence may be direct (by the effect of the considered change in $u$ on the calculation of the output) or indirect (by the effect of past $y$'s, themselves influenced by the considered change). The global influence of a change of the control input on one future output will be the sum of both direct and indirect influences.

We have defined $l$ as the number of past and present $y$'s and $(m+1)$ the number of past and present $u$'s needed to define the predicted output. Then $\partial\hat{y}(n+i)/\partial u(n+k)$ can be written as:

$$\frac{\partial\hat{y}(n+i)}{\partial u(n+k)} = \sum_{s=1}^{l+m+1}\left(\frac{\partial\hat{y}(n+i)}{\partial X_s^{(0)}}\cdot\frac{\partial X_s^{(0)}}{\partial u(n+k)}\right), \tag{5.25}$$

where the elements $X_s^{(0)}$, $s = 1, \ldots, l+m+1$, form the inputs of the neural network used to calculated $\hat{y}(n+i)$. Let's illustrate first this calculation with a simple example in which the neural network is linked to itself only once (this means the system has a prediction horizon of two sampling times in the future), as shown in Figure 5.6. The calculation of $\partial\hat{y}(n+d)/\partial u(n)$ is simple, because $u(n)$ is a direct input to the first neural network. However, $\partial\hat{y}(n+d+1)/\partial u(n)$ is a function of $u(n)$ directly and indirectly through $\hat{y}(n+d)$. If the second neural network is considered, it is clear that the influence that is to be determined is active only through $\hat{y}(n+d)$ and $u(n)$. The term $\partial\hat{y}(n+d+1)/\partial u(n)$ can be written as follows:

$$\frac{\partial\hat{y}(n+d+1)}{\partial u(n)} = \frac{\partial\hat{y}(n+d+1)}{\partial X_{l+2}^{(0)}} + \frac{\partial\hat{y}(n+d+1)}{\partial X_1^{(0)}}\cdot\frac{\partial X_1^{(0)}}{\partial u(n)} \tag{5.26}$$

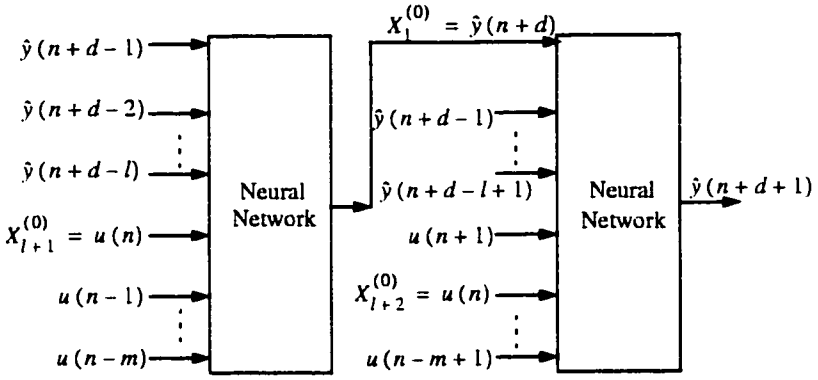



**FIGURE 5.6. Example neural network for recursive multi-step ahead prediction.**

The general algorithm for computing $\partial\hat{y}(n+i)/\partial u(n+k)$, $d \le i \le L, 0 \le k \le N-1$ can now be described and is given in Figure 5.7. In the numerical studies considered in this chapter, we limit the prediction horizon to $L = d+1$, and the control horizon to $N = L-d+1 = 2$. As a result, we only need to find the following dynamic sensitivity derivatives: $\partial\hat{y}(n+d)/\partial u(n)$, $\partial\hat{y}(n+d+1)/\partial u(n)$ and $\partial\hat{y}(n+d+1)/\partial u(n+1)$.

Recursive Predictor:

Algorithm for computing $\partial \hat{y}(n+i)/\partial u(n+k)$ , $d \le i \le L$, $0 \le k \le L-d$:

for $i = 1$ to $d-1$

Predict $\hat{y}(n+i)$ using

$X^{(0)} = [\hat{y}(n+i-1), \ldots, \hat{y}(n+i-l), u(n+i-d), \ldots, u(n+i-d-m)]^T$

as input to the neural network. Note that if $i-k \le 0$, $k = 1, 2, \ldots, l$, then

$\hat{y}(n+i-k) = y(n+i-k)$ .

endfor($i$)

for $j = d$ to $L$

Predict $\hat{y}(n+j)$ .

Compute $\delta_j^{(p)}$ , $j = 1, 2, \ldots, m_{(p)}$ , starting from $p = p_t$, then

backpropagate in neural network recursively to obtain $\delta_j^{(p)}$ for

$p = p_t - 1, p_t - 2, \ldots, 2, 1$ .

for $k = 0$ to $j-d$

Compute the **direct** influence of $u(n+k)$ on $\hat{y}(n+j)$

$$\frac{\partial \hat{y}(n+j)}{\partial u(n+k)} = \sum_{j=1}^{m_{(1)}} \delta_j^{(1)} w_{ji}^{(1)} ,$$

where $i$ in the above equation denotes the input node $u(n+k)$.

Note: The above sensitivity derivatives can be calculated in parallel for each $k$.

Compute the **direct** and **indirect** influence of $u(n+k)$ on $\hat{y}(n+j)$

$$\frac{\partial \hat{y}(n+j)}{\partial u(n+k)} = \sum_{s=1}^{l+m+1} \left( \frac{\partial \hat{y}(n+j)}{\partial X_s^{(0)}} \cdot \frac{\partial X_s^{(0)}}{\partial u(n+k)} \right) .$$

endfor($k$)

endfor($j$)

**FIGURE 5.7. Algorithm for computing the dynamic sensitivity derivatives for the recursive $L$-step ahead predictor.**

### 5.4.2. Non-Recursive (Direct) Multi-step Ahead Predictor

Compared with the recursive $i$-step ahead predictor, the non-recursive predictor is simple for $i$-step ahead prediction as shown in Figure 5.8, since it does not need the recursive procedure. In this case, we use a single neural network with multiple outputs. For the direct $L$-step ahead predictor, the dynamic sensitivity derivatives $\partial\hat{y}(n+i)/\partial u(n+k)$, $d \le i \le L$, $0 \le k \le N-1$ (where we select $N = L-d+1$) required in Eq. (5.14) can be computed from the algorithm given in Figure 5.9 using the gradient functions defined by the algorithm in Figure 5.5. The calculation of the dynamic sensitivity derivatives allow a parallel implementation, and as a result, fast computation times are possible.

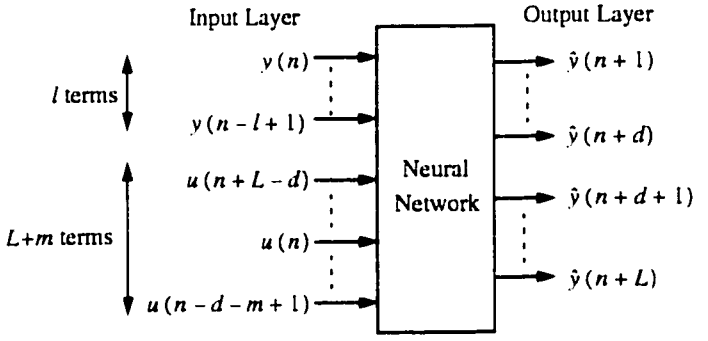



**FIGURE 5.8. Direct $L$-step ahead prediction.**

Non-Recursive (Direct) Predictor:

Algorithm for computing $\partial \hat{y}(n+i)/\partial u(n+k)$ , $d \le i \le L$, $0 \le k \le L-d$:

Define local gradient for layer $p$, neuron $i$:

$$\delta_i^{(p)} = \begin{cases} g\left(z_i^{(p)}\right), & \text{if } p = p_t \\ \sum_{j=1}^{m_{(p+1)}} \left(\delta_j^{(p+1)} w_{ji}^{(p+1)}\right) g\left(z_i^{(p)}\right), & \text{if } 1 \le p < p_t, \end{cases}$$

where the input layer is given by

$$X^{(0)} = [y(n), \ldots, y(n-l+1), u(n+L-d), \ldots, u(n), \ldots, u(n-d-m+1)]^T.$$

Computation Steps:

for $s = d$ to $L$

Compute $\delta_j^{(p)}$ , $j = 1, 2, \ldots, m_{(p)}$ , starting from $p = p_t$ for output $\hat{y}(n+s)$ , then

backpropagate in neural network recursively to obtain $\delta_j^{(p)}$ for $p = p_t - 1, p_t - 2, \ldots, 2, 1$ .

for $k = 0$ to $s-d$

$$\frac{\partial \hat{y}(n+s)}{\partial u(n+k)} = \sum_{j=1}^{m_{(1)}} \delta_j^{(1)} w_{ji}^{(1)} ,$$

where $i$ in the above equation denotes the input node $u(n+k)$.

endfor($k$)

endfor($s$)

Note: The dynamic sensitivity derivatives can be calculated in parallel.

**FIGURE 5.9. Algorithm for computing the dynamic sensitivity derivatives for the direct $L$-step ahead predictor.**

## 5.5. Neural Prediction Example

To illustrate the neural predictive schemes, we use an M/M/1 dynamic queueing model with the first-order approximation described by the difference equation

$y(n+1) = y^2(n)/(1+y(n)) + u(n)$ , with the initial condition $y(0) = y_0$, where $y(n)$ and $u(n)$ represent the time-dependent queue size and arrival rate, respectively [Fili88]. The time units are normalized in such a way that the service capacity is equal to unity. A sample arrival rate pattern is used to illustrate the prediction schemes: a sinusoidal pattern given by $u(n) = (1 + \sin(2\pi n/T))/2.5$, where $T$ is the simulation time as shown in Figure 5.10. The initial queue size is assumed to be $y(0) = 0$. The input $u(n)$ and output $y(n)$ responses used for the neural network training are shown in Figure 5.10.

To model the response shown in Figure 5.10, the recursive and non-recursive predictors are used. For the results given below, the following neural architectures were used: 1). Recursive Predictor: 16 inputs, (8 $y$ terms and 8 $u$ terms), 3 layers of sigmoidal neurons (16x16x1), 1 output; 2). Non-Recursive Predictor: 22 inputs (8 $y$ terms and 14 $u$ terms), 3 layers of sigmoidal neurons (22x22x10), 10 outputs. The system output $y$ was predicted 5 and 10 time steps into the future. The input to the neural network predictor consists of a moving window of $u(n)$ and $y(n)$ values as illustrated in Figure 5.11. Past and present values of $y$ and $u$ as well as future values of $u$ are fed to the neural network. The output from the neural network is the output $y$ in the future. It should be emphasized here that since the training data is synthetically generated, the future $u$ values (relative to the center of the window) are known and these can be used for training. Section 5.3 which describes the predictive congestion control algorithm discusses how to calculate the future $u$ values for a control problem.

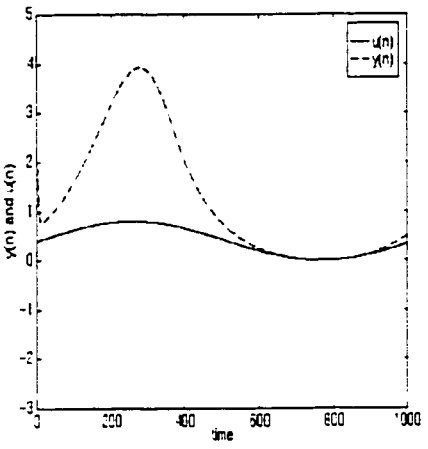



**FIGURE 5.10. Example training data for neural prediction.**

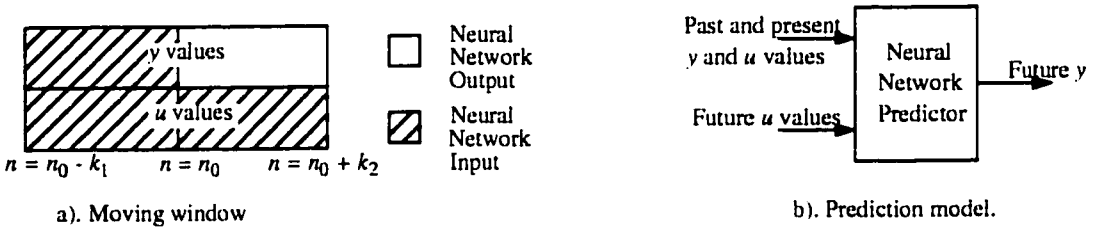



**FIGURE 5.11. Prediction scheme.**

Figure 5.12 (for the recursive prediction) and Figure 5.13 (for the non-recursive prediction) illustrate the neural predictive models' ability to learn the nonlinear process characteristics described above. Since the neural networks weights are initially randomized, the neural network does not do a very good job in its prediction at first. After a small number of steps training, the neural network gives excellent predictions, as the results show. In the studies, we found that the agreement between predicted and actual $y$ values at the 5th and even in most cases the 10th step can be as good as (or close to) that at the one time step. Two points should be noted in the above prediction examples. First, in the prediction

examples the data is completely known. Thus, the future values of $u$ are known for training. Second, in using a neural network for on-line control, the future $u$ values are unknown but they can be calculated as optimization variables by the control algorithm described earlier.

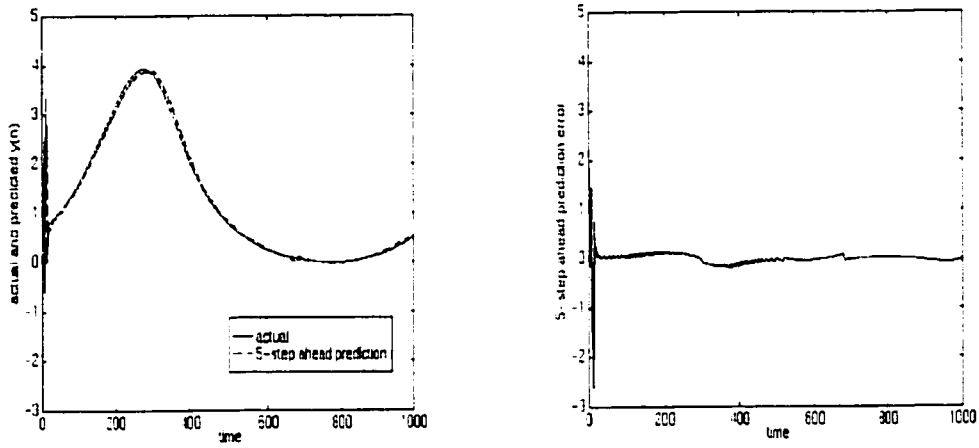



**FIGURE 5.12. Prediction of test data: Recursive neural prediction.**

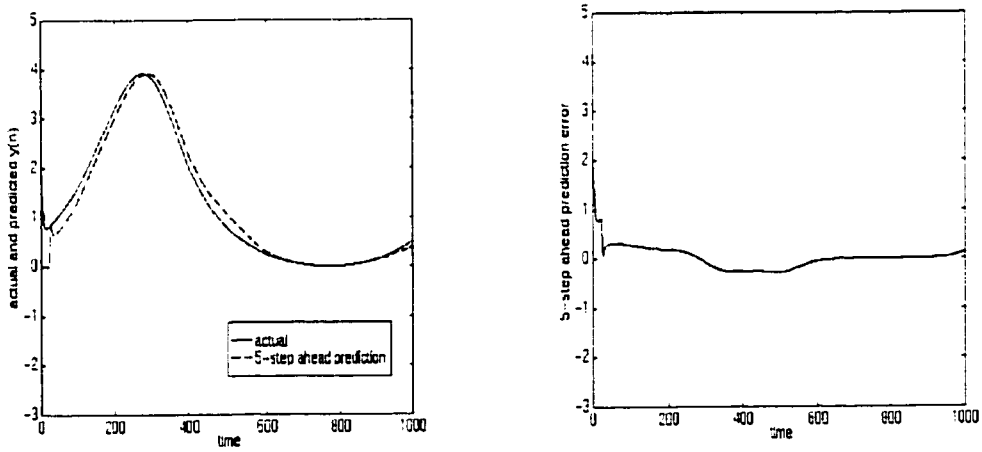



**FIGURE 5.13. Prediction of test data: Direct neural prediction.**

## 5.6. Adaptive Predictive Control of Queueing Systems

In the previous sections, we developed a feedback-based congestion control scheme using the predictive capabilities of neural networks and a gradient descent technique. The scheme is based on a control algorithm running at a bottleneck network node which continually collects state information, evaluates the available bandwidth for the sources and transmits this information to the sources using control packets. In this section, we describe the control procedures for multiple sources transmitting data to a common bottleneck node buffer. A control algorithm running at the bottleneck node evaluates the resource need of each source and distributes the estimated available resources accordingly. Several techniques can be used for the resource distribution, a simple mechanism is one in which the available bandwidth is equally distributed among the sources. But this technique can lead to the situation where some of the sources will not use their share of the bandwidth because they do not need all the bandwidth they are granted. What is needed are mechanisms which reduce resource waste. By adjusting the amount of bandwidth that is distributed and varying the way it is shared among the sources, the controller achieves a tight control of the whole system.

We consider systems in which multiple sources transmit data without resource reservation to a common bottleneck buffer with information transfer delays between the sources and the bottleneck buffer. If at a given instant several sources try to transmit data simultaneously to the buffer, the rate of the superposed flow can exceed the available service rate of the buffer if the source rates are not properly controlled. If this situation lasts for an extended period of time, the destination buffer will build up and eventually overflow and

can lead to an overload situation in the network. This converging flows problem can be alleviated by the use of a feedback control mechanism that coordinates the emission of the sources as described in the previous sections. Also, if the goal is to achieve anything better than a simple "best-effort" policy, a closed-loop control between the sources and the destination is necessary in order to coordinate the data emissions of the sources.

The control problem under study is modeled as a queueing system as depicted in Figure 5.14. It consists of $M$ sources transmitting data towards a single bottleneck node with control loop delays between each source and the bottleneck node. In order to control the data flows and allocate the network resources, the network nodes exchange information and control packets besides the users data packets.

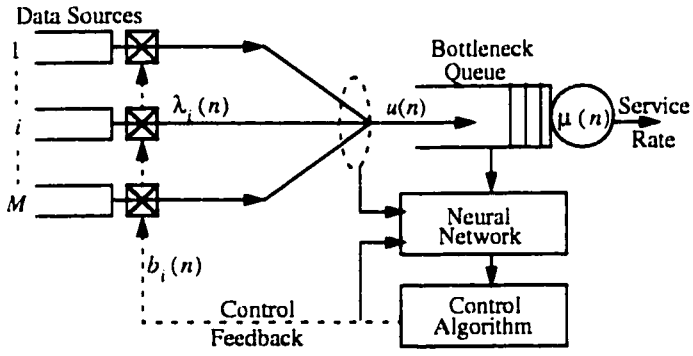



**FIGURE 5.14. Congestion control model with multiple sources.**

It is clear that the key point of this control architecture lies in the control algorithm that is employed at the bottleneck node. The control algorithm can be divided into two distinct phases:

- The bottleneck node must evaluate the available resources that are at its disposal at a given instant. In our model, this consists of computing the quantity $u(n)$, the amount of available bandwidth at time $n$. This quantity varies with the occupancy of the bottleneck node buffer and with the effective rate at which data departs from this buffer.
- The bottleneck node must share those resources between the different sources, according to a policy. This consists of computing for each source the quantity $\sigma_i(n)$ which is the best share of the available resources to be attributed to source number $i$ ( $1 \leq i \leq M$, $\sum_{i=1}^{M} \sigma_i(n) = 1$ ). Then the feedback control signal to source $i$ can be expressed as:

$$b_i(n) = \sigma_i(n)\, u(n) . \tag{5.27}$$

Now the problem consists of determining good choices for the functions $\sigma_i$ and $u$. A resource sharing function can be used to evaluate the amount of resources (in terms of bandwidth) to be given to each source, according to a predefined policy. A control packet indicating the service rate $b_i(n)$ that source $i$ should use is sent at regular intervals to each source. Upon reception, the source sets its transmission rate $\lambda_i(n)$ according to the value specified in the control packet.

We propose the on-line congestion control and neural network training algorithm given in Figure 5.15. In the case of multiple sources transmitting data to the bottleneck node, the control loop delay parameter $d$ in the control algorithm is set to a single representative value, for example, $d = min(d_1, d_2, \ldots, d_M)$, where $d_i$, $i = 1, 2, \ldots, M$, are the individual loop delays of each source transmitting data to the bottleneck buffer.

At time instant $n$:

Step 1: Get measurement of the system output $y(n)$ (which corresponds to input $u(n-d)$ ).

Available system history maintained by the controller:

Past system outputs: $y(n-1), \ldots, y(n-l+1)$ .

Past controller outputs: $u(n-1), \ldots, u(n-d-m+1)$ .

Controller outputs computed in previous control interval:

$$U(n) = [u(n), u(n+1), \ldots, u(n+L-d)]\ .$$

Step 2: Predict the system output over the range of future times.

i). Recursive Multi-Step Ahead Predictor: Predict $\hat{y}(n+i)$ for $1 \le i \le d-1$ using

$$X^{(0)} = [\hat{y}(n+i-1), \ldots, \hat{y}(n+i-l), u(n+i-d), \ldots, u(n+i-d-m)]^T$$

as neural network input. These predictions are to be used to predict the next set $\hat{y}(n+i)$ , $d \le i \le L$.

Predict $\hat{y}(n+i)$ and then compute $\partial \hat{y}(n+i)/\partial u(n+k)$ , for $d \le i \le L$, $0 \le k \le L-d$.

ii). Direct Multi-Step Ahead Predictor: Predict $\hat{y}(n+i)$ , $1 \le i \le L$ using

$$X^{(0)} = [y(n), \ldots, y(n-l+1), u(n+L-d), \ldots, u(n), \ldots, u(n-d-m+1)]^T$$

as neural network input. Then compute $\partial \hat{y}(n+i)/\partial u(n+k)$ , for $d \le i \le L$, $0 \le k \le L-d$.

Step 3: Compute gradient function $\nabla J(n) = \partial J/\partial U(n)$ .

Compute the control inputs $U(n+1) = U(n) - \eta \dfrac{\partial J}{\partial U(n)}$ .

Use the first element of $U(n+1)$ as the current control input.

Step 4: Training of neural network:

i). Recursive Multi-Step Ahead Predictor:

Use $[\hat{y}(n) - y(n)]$ as neural network error to update neural network weights.

$$\hat{y}(n) = \hat{f}[y(n-1), \ldots, y(n-l), u(n-d), \ldots, u(n-d-m)]\ .$$

ii). Direct Multi-Step Ahead Predictor:

Use $\displaystyle\sum_{k=n-L+1}^{n} [\hat{y}(k) - y(k)]$ as neural network error to update neural network weights

Step 5: Next interval, $n \Leftarrow n+1$ , go to Step 1.

**FIGURE 5.15. Algorithm for on-line congestion control and neural network training at the bottleneck network node.**

## 5.7. The Resource Sharing Function

The allocated rate $b_i(n)$ for a data source $i$ is computed using $b_i(n) = \sigma_i(n)\, u(n)$, where $\sigma_i(n)$ is a resource sharing or fair share function, and $u(n)$ the total available bandwidth for the low priority traffic. Here we define a function $\nu$ so that, if $x$ represents the state of a source (e.g., its current queue size or data rate), then the dimensionless quantity $\nu(x)$ will represent a measure of its need of resources. Then the available resources at a given instant will be shared between the transmitting sources proportional to this quantity. This can be written as:

$$\sigma_i(n) = \frac{\nu_i(x)}{\sum_{j=1}^{M} \nu_j(x)}, \quad i = 1, 2, \ldots, M. \tag{5.28}$$

A simple control mechanism, called the *equal sharing* policy, would be to share equally among the sources the bandwidth available at the bottleneck node. A simple choice for $\sigma_i(n)$ is to record the number $M(n)$ of low priority connections going through the output link and to set $\sigma_i(n) = 1/M(n)$. But this technique can lead to the situation where some of the sources will not be using their share of the bandwidth because they do not need all the bandwidth they are granted. What is needed are policies which reduce resource waste. To catch the bandwidth left available by other sources that do not use their fair share, the following function can be introduced [More97]

$$\sigma_i(n) = (1 - \sigma_0)\, \rho_i(n) + \sigma_0, \tag{5.29}$$

where $\rho_i(n)$ is the bandwidth ratio of connection $i$ and $\sigma_0$ the minimal value of the function. For example, if $\sigma_0 = 0$, then $\rho_i(n)$ can be evaluated proportional to the current rate $\lambda_i(n)$ of connection $i$, and the total current rate $\lambda_t(n)$, assuming the source and the network node are $T$ time units apart, i.e.,

$$\rho_i(n) = \frac{\lambda_i(n-T)}{\lambda_t(n-T)}. \tag{5.30}$$

If per-connection queue size accounting is implemented in the node, the above function can be modified to use the state information available for each individual connection. Then, we can define

$$\rho_i(n) = \frac{y_i(n)}{y_t(n)}, \tag{5.31}$$

where $y_i(n)$ is the number of packets in the low priority queue belonging to connection $i$ and $y_t(n)$ is the total number of packets in the queue.

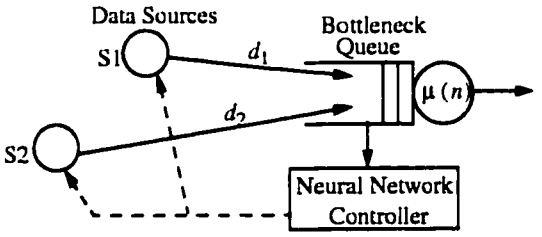
## 5.8. Numerical Results

In the simulation experiments, initial occupancy of the bottleneck buffer, $y(0)$, and the input rates for the first $d_i$ sampling intervals ($\lambda_i(1), \lambda_i(2), \ldots, \lambda_i(d_i)$) are initialized for each source $i$. At each sampling interval, $n$, inputs to the buffer and outputs from the buffer are generated using the control inputs computed by the controller and the queue service rate, and the buffer occupancy is updated. To limit the size of queue buildup during the

start of a connection, we assume that a source initially transmits at a (small) specified rate until it receives its first control packet. We assume that a procedure for the selection of the initial transfer rate is defined. We also assume in our study that the traffic sources are cooperative in the sense that they transmit at the controlled rate and keep any excess traffic at the source.

We consider the case of unequal loop delays between the sources and the bottleneck node as shown in Figure 5.16. For unequal loop delays, we evaluate the control system performance for two cases:

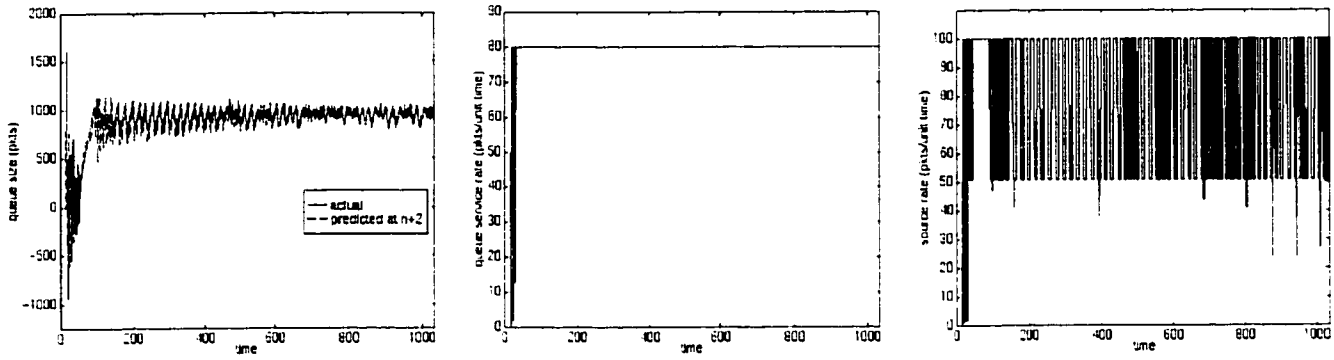
- The value of the control loop delay parameter $d$ used in the control algorithm is set to the round trip delay of the nearest source, $d = min(d_1, d_2, ..., d_M)$, where $M$ is the total number of sources transmitting data to the bottleneck node.
- The value of the control loop delay parameter $d$ used in the control algorithm is set to the round trip delay of the farthest source, $d = max(d_1, d_2, ..., d_M)$.



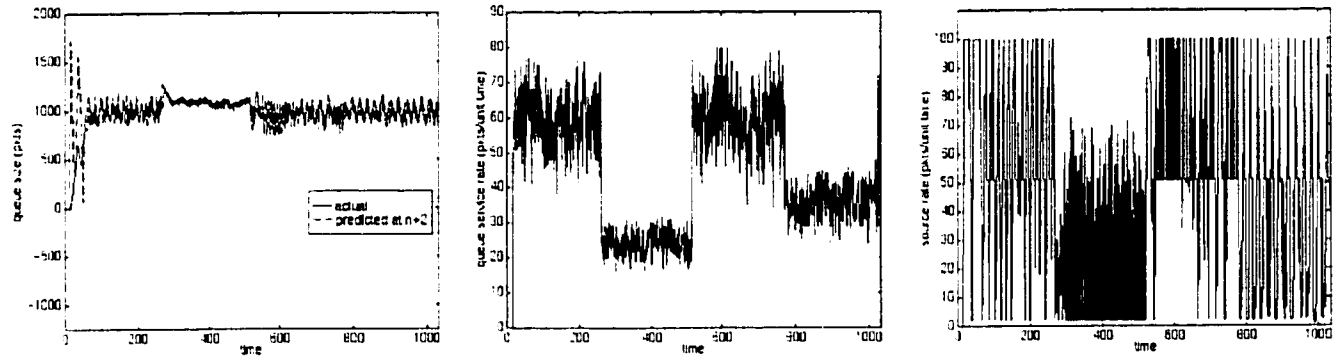

1). Control loop delay $d = d_1$, $d_1 < d_2$
2). Control loop delay $d = d_2$, $d_1 < d_2$
b).

**FIGURE 5.16. Predictive control of multiple sources: A single bottleneck network node configuration.**

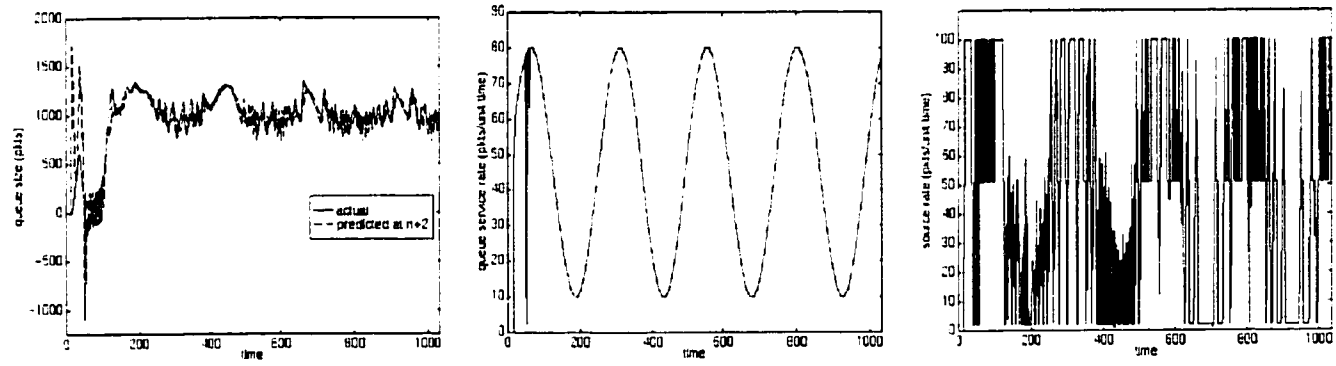
In our numerical studies, we assume that the sources are persistent (always with data to transmit) and a simple resource sharing policy, that is, the bottleneck network node shares equally the available bandwidth among the sources at the node. The congestion threshold $r$ is set at $r = 1000$ packets. We consider a round trip delay of 2 time units (time unit = 0.25ms) for the nearest source (source 1) and 6 time units for the farthest source (source 2). These two cases result in $d = 2$ and $d = 6$. Three bottleneck queue service rate patterns are assumed in the studies: constant (80 packets/time unit), sinusoidal ($integer\,[35\,(1 + \sin\,(10\pi n/T)) + 10]$ packets/time unit, where $T$ is the simulation time), and random (maximum = 80 packets/time unit). A direct multi-step neural predictive architecture is used: 3 layer neural network ($L + m + l$ inputs, $L + m + l$ input neurons, $L + m + l$ hidden neurons, $L$ output neurons); $l = 8$ terms of $y$; and $L + m$ terms of $u$ ($m = 5$). The value of the control loop delay in the control algorithm is $d$, the prediction horizon $L = d + 1$, and the control horizon $N = L - d + 1$. The results of the simulations for a single bottleneck queue are given in Figures 5.17 to 5.20. The results show that the neural predictive congestion control technique is able to maintain the queue size close to the threshold of 1000 packets for the three queue service rate patterns studied. The performance of the system is also good for time varying queue service rates. Setting the control loop delay $d$ to 2 (Figure 5.17) was found to give better system performance than setting $d$ to 6 (Figure 5.19), i.e., $d = min\,(d_1, d_2, \ldots, d_M)$ gives the best system performance. It was also noticed that the neural training was relatively difficult for the case of $d = 6$.




a). Bottleneck queue with a constant service rate.

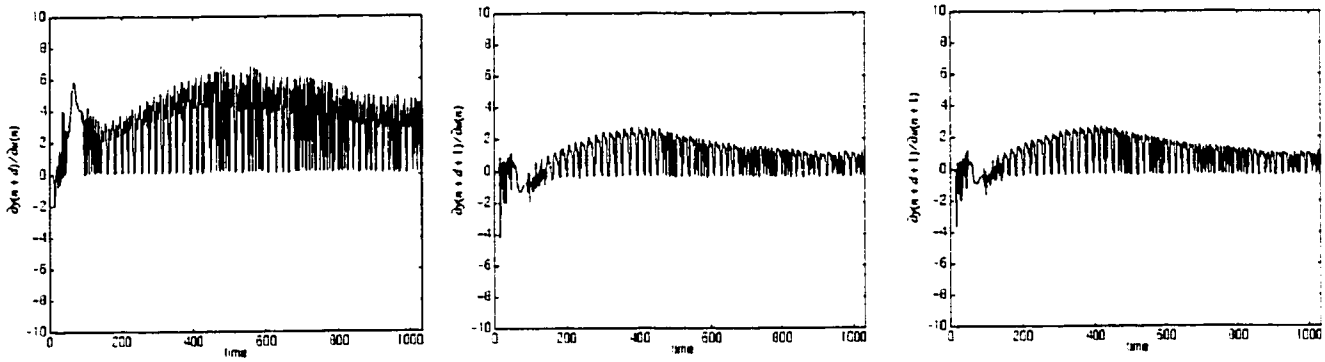



b). Bottleneck queue with a "random" service rate.

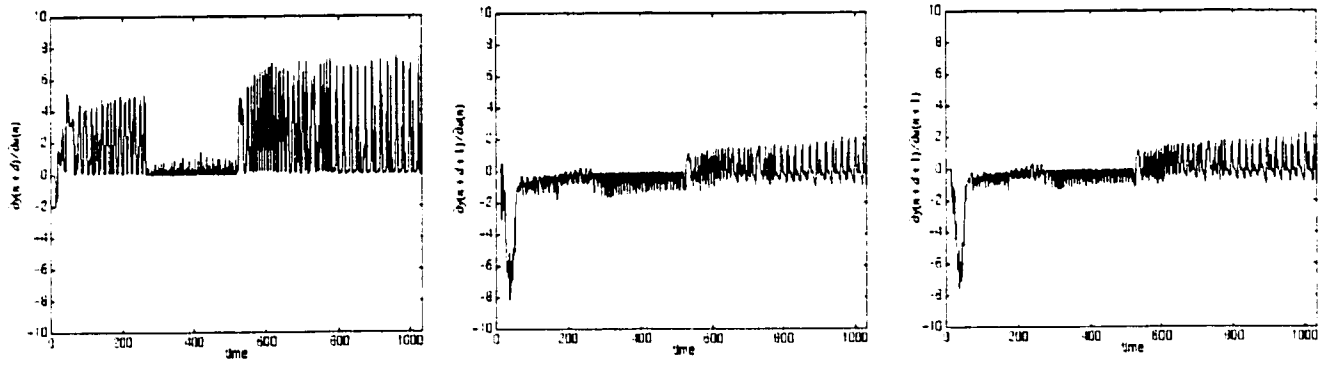



c). Bottleneck queue with a "sinusoidal" service rate.

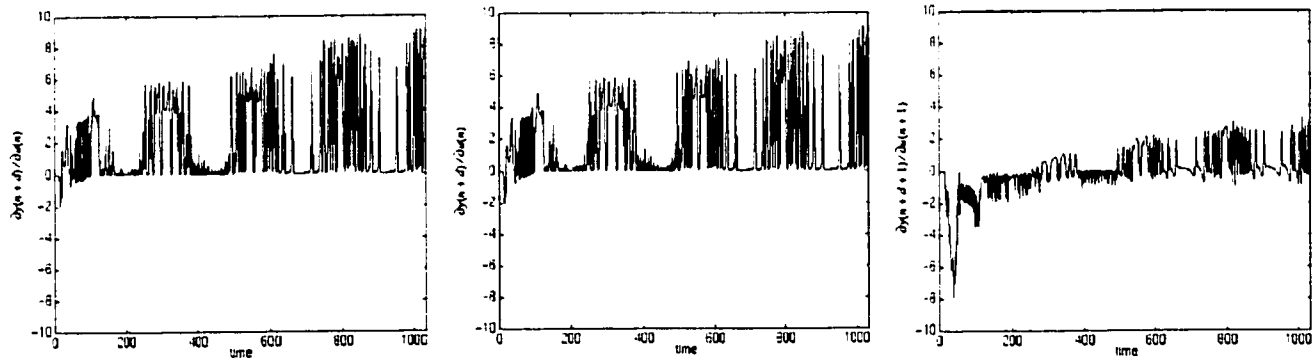
**FIGURE 5.17. Evolution of system states with time: Control algorithm delay parameter set to the smallest round-trip delay.**



a). Dynamic sensitivity derivatives for the bottleneck queue with a constant service rate.

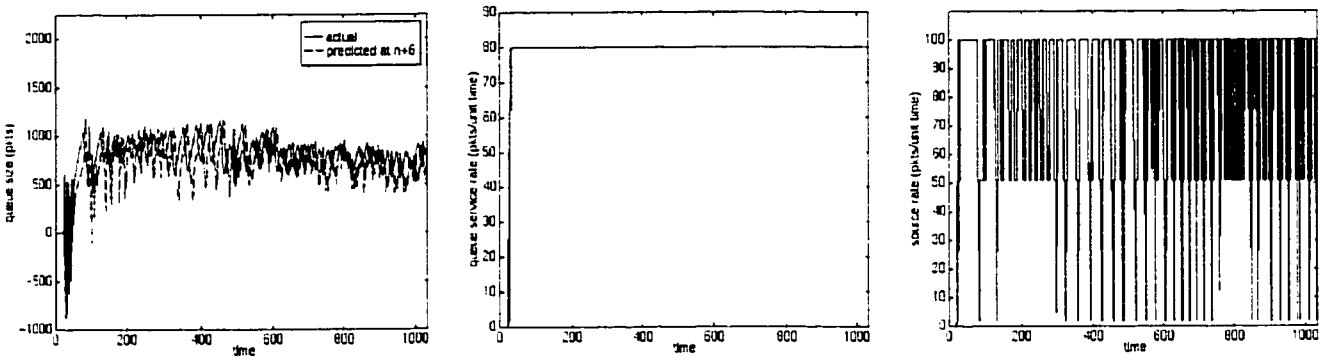


b). Dynamic sensitivity derivatives for the bottleneck queue with a "random" service rate.

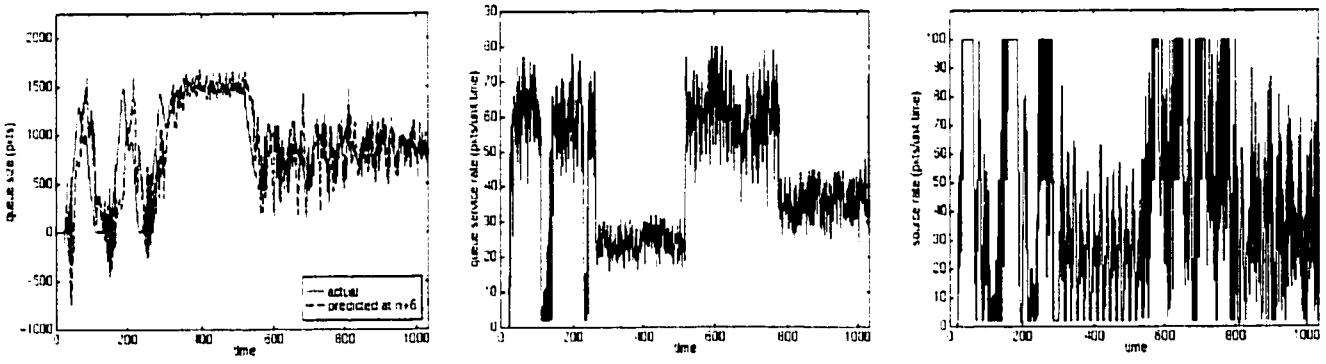


c). Dynamic sensitivity derivatives for the bottleneck queue with a "sinusoidal" service rate.

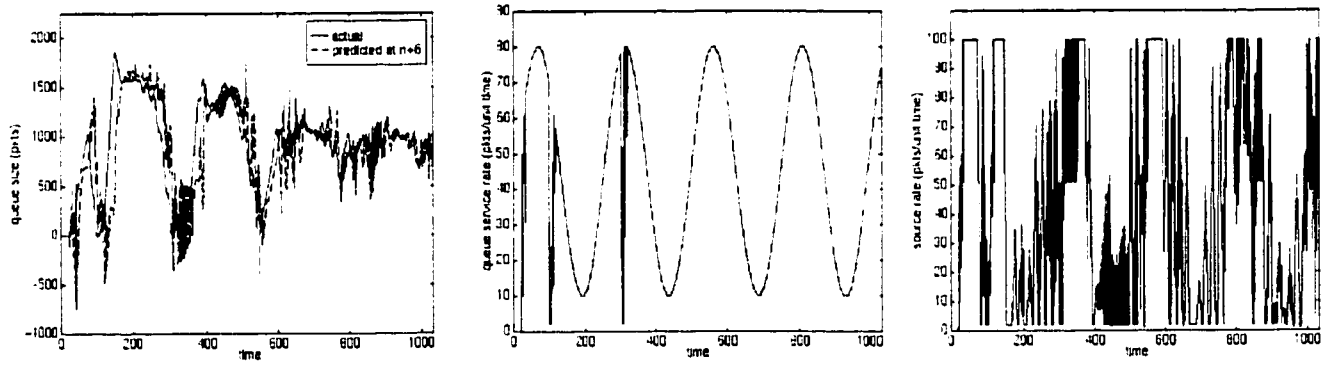
**FIGURE 5.18. Dynamic sensitivity derivatives: Control algorithm delay parameter set to the smallest round-trip delay.**




a). Bottleneck queue with a constant service rate.

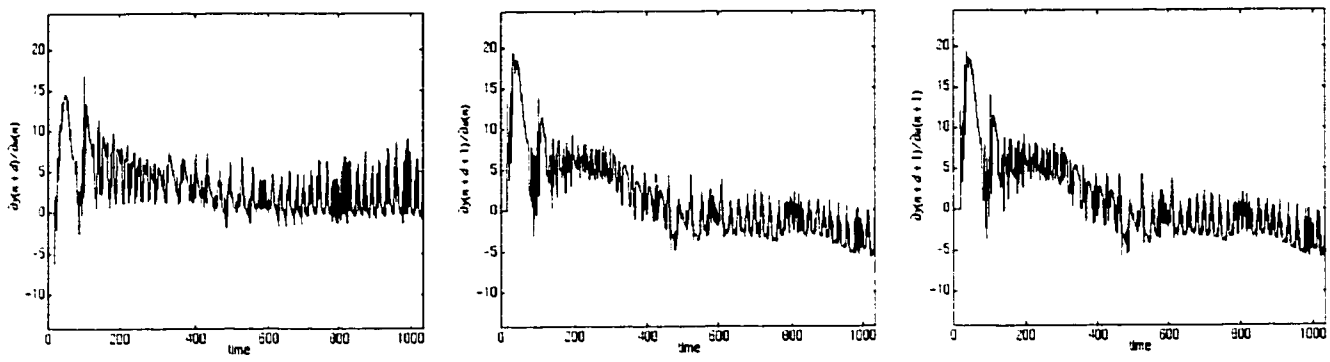



b). Bottleneck queue with a "random" service rate.

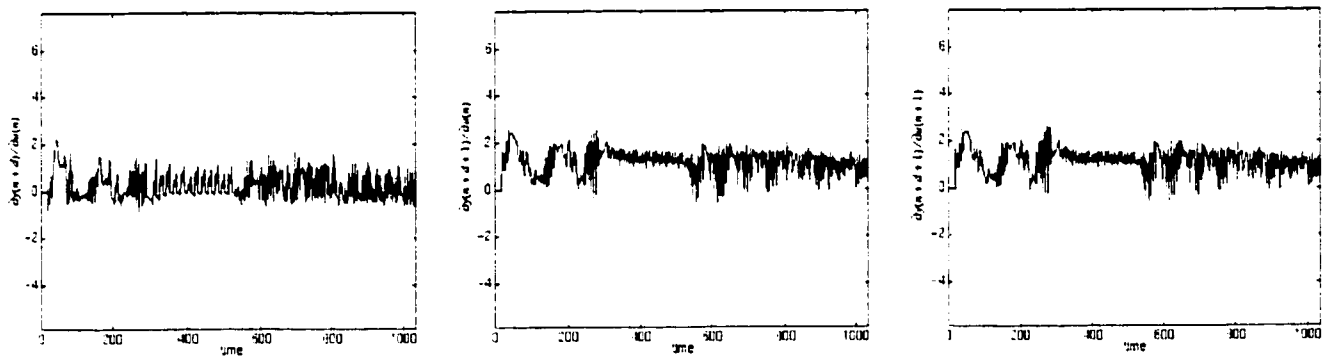



c). Bottleneck queue with a "sinusoidal" service rate.

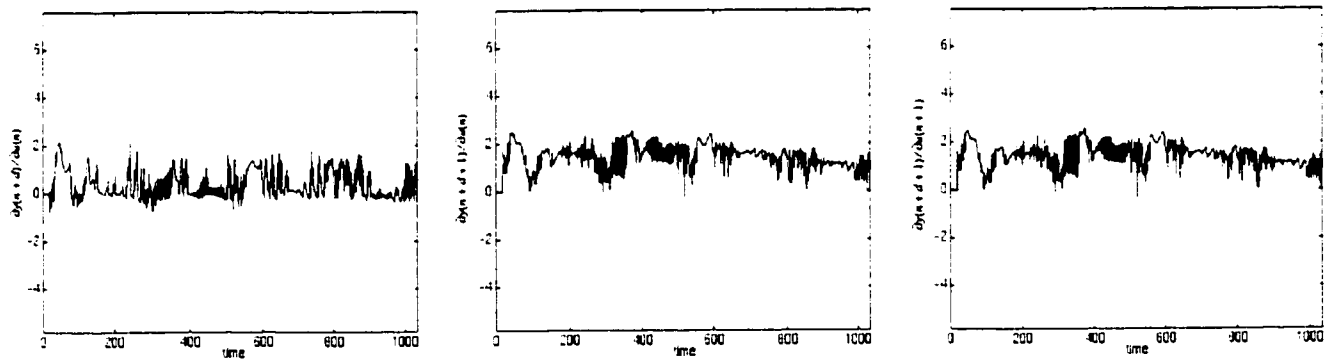
**FIGURE 5.19. Evolution of system states with time: Control algorithm delay parameter set to the largest round-trip delay.**



a). Dynamic sensitivity derivatives for the bottleneck queue with a constant service rate.



b). Dynamic sensitivity derivatives for the bottleneck queue with a "random" service rate.



c). Dynamic sensitivity derivatives for the bottleneck queue with a "sinusoidal" service rate.

**FIGURE 5.20. Dynamic sensitivity derivatives: Control algorithm delay parameter set to the largest round-trip delay.**

## 5.9. Conclusion

We have presented in this chapter, a technique that employs neural dynamic models in a predictive congestion control scheme. Predictive models which account for the control loop delays in the data transfer process are developed and procedures for the control signal computations are given using gradient functions of the neural network models. Two neural architectures for predicting the system output are considered. The first model employs a recursive predictive approach where the neural network output is linked back to itself to go as far as needed in the prediction horizon. The second model is one in which all the predictions are made at the same time at different nodes at the output layer of the neural network.

The development in this chapter (and similarly in the previous chapters) is motivated and constrained by ideas that are familiar in systems theory and adaptive control. This allows us to make parallels and use insights developed in these fields. The theory is based on some heuristics, it is not backed up by the type of convergence and stability analysis which is commonly applied in the field of adaptive control. However, it becomes apparent that the neural network modeling approach provides a conceptual basis and a procedure for nonlinear prediction and control. Some issues that arise in the application of adaptive control are stabilizability and solvability of the control model. Neither of these issues have been completely resolved in the linear adaptive theory and it is likely that the nonlinear problem is going to be harder to deal with.

The simulation studies have given us a further understanding of the capabilities and limitations of the proposed neural predictive control techniques. Since the control scheme relies on a good neural network architecture and training algorithm to compute the control sig-

nals to the data sources, studies may be needed to determine neural architectures and training algorithms that can adapt well to the dynamic conditions in the data transfer process. The aim of the chapter has been to demonstrate via examples how neural network modeling may be used for network congestion prediction and control.

In all the problems studied we found that the algorithm converged to an acceptable solution. However, some areas of further investigation are: 1). how far can the predictions go before there are significant prediction errors, 2). control models using other cost functions, and 3). investigate other neural network architectures and training schemes apart from the backpropagation algorithm used in the current studies. To conclude, we note that numerous question remain unanswered, however, we believe that the work described here has made significant contributions in providing congestion control solutions based on system sensitivities and adaptive control.

# Chapter 6: Conclusions and Suggestions for Further Research

## 6.1. Main Results

This thesis addresses the modeling and performance evaluation of congestion control techniques based on adaptive control and sensitivity analysis. In this chapter, we review the results from the preceding chapters, and examine the extent to which we have been successful in our research effort. We also examine the contributions and weaknesses in our work, and areas for further research.

The main conclusions of this research are now presented. It was shown in Chapter 2 that the application of sensitivity analysis can provide useful information to system control and management functions in determining the cause of deviation of system performance and provide a controlling basis for them in order to operate the system more efficiently. Sensitivity analysis is performed by taking the partial derivative of the system performance with respect to each individual input to obtain a sensitivity coefficient. These coefficients can then be used as a basis for inference about input-output relationships in the system and they can also easily support efficient implementations of gradient-based methods for process optimization as described in Chapter 3. In the congestion control framework described in Chapter 3, system monitoring and sensitivity analysis allow for the optimization of the system performance. The congestion controller is provided with a form of sensitivity analysis that gives instantaneous estimates of sensitivity coefficients. Such a feature used in

conjunction with a trained neural network model with real-time inputs provides a basis for automatic adjustment of system input rates to optimize performance.

In Chapter 4, it was shown that significant improvements in system performance can be obtained in a binary feedback congestion control scheme when the setting of the single bit congestion indication communicated back to the data sources by the network nodes is based upon the sign of the sensitivity of a system performance function. Congestion bit setting based on queue length is the commonly used binary feedback congestion control approach. The sign of the performance sensitivity function gives the optimal direction for the data source rate adjustment which can be done in an additive increase/multiplicative decrease manner. The proposed scheme uses a simple neural network model of the system dynamics to determine the performance sensitivity function. The studies in Chapter 4 showed that the sensitivity-based scheme has better performance compared to the conventional queue thresholding scheme. The sensitivity-based scheme has smaller oscillations in queue size and source rates, and the oscillations do not vary significantly even if there are large changes in bottleneck queue service rates.

The sensitivity-based approach for congestion detection and bit setting is not intended to solve the limitations of single-bit binary congestion control. A problem with single-bit congestion control schemes is that they generally result in large oscillations in queue size. The main reason is that the single congestion bit can indicate only the existence of congestion somewhere along the data connection, but not the location or severity. It also takes a complete round-trip before the congestion information returns to the source end-system. In addition, most implementations of the basic binary congestion control scheme in switches consider a common shared buffer and the service discipline of First-Come-First-Served

(FCFS). However, a congested switch has no ability to mark sources selectively. In particular, a switch will mark all sources regardless of the individual source's rates. This gives rise to the "beat down" problem, where a connection traversing multiple hops is subject to being marked more often than those traversing fewer congested links (owing to the unsynchronized congestion waves at each of these switches), with the result that its bandwidth is driven down for unacceptably long periods of time. This undesirable effect is proportional to the number of traversed congested links. The limitation of non-selective binary marking mechanisms implies the need for some form of intelligent marking to achieve fair distribution of available bandwidth [RobeL94][SiuK96].

The studies in Chapter 4 show that the sensitivity-based approach is able to reduce the magnitude of the queue oscillations but cannot completely eliminate the oscillations. Using the sensitivity of the system performance function makes it possible to detect congestion early and this leads to early feedback control reaction by the data sources. However, the sensitivity-based approach is dependent on finding neural architectures and training schemes suitable for modeling the dynamics of the network queues. The neural architecture used in the studies is a simple one and it is believed that even better performance can be obtained if good architectures capable of modeling the system dynamics can be obtained.

In feedback-based congestion control schemes, information transfer delays make the rate control signals received at the data sources or the network access points from the network outdated. The congestion control scheme described in Chapter 5 employs a neural network to predict the state of congestion in a computer network over a prediction horizon. Analytic procedures for the source rate control signal computations were given using sensitiv-

ity functions of the neural predictor. The studies indicate that the multi-step neural predictive technique for congestion control provides good control adaptability and performance. The development in this chapter (and also in Chapter 3) is motivated and constrained by ideas that are familiar in systems theory and adaptive control. This allows us to make parallels and use insights developed in these fields. The theory is based on some heuristics, it is not backed up by the type of convergence and stability analysis which is commonly applied in the field of linear adaptive control. The formalism is somewhat loose and a mathematical foundation is presently lacking as is the case with most neural control problems. However, it becomes apparent that the neural network approach and sensitivity analysis provides a conceptual basis and a procedure for nonlinear prediction and control. Most notably, the major issues that arise in the application of adaptive control are stabilizability and solvability of the control model. Neither of these issues have been completely resolved in the linear adaptive theory and it is likely that the nonlinear problem is going to be harder to deal with.

This thesis has made a number of contributions to the area of reactive congestion control. We review some of them here.

- This thesis has provided basic information on how neural network models can be used to represent sensitivity functions for the purpose of performance monitoring and control of queueing systems. The sensitivity coefficients can be used as a basis for inference about input-output relationships in the system and they can also easily support efficient implementations of gradient-based methods for process optimization (i.e., control of congestion).

- The thesis has presented a new binary feedback congestion control scheme in which the setting of the single bit congestion indication communicated back to the data sources by the network nodes is based upon the sign of the sensitivity of a system performance function. The scheme employs a neural network model of the system dynamics to determine the performance sensitivity function.

- Another contribution is the development of a multi-step adaptive predictive technique for the control of congestion in a network queue in which the prediction is made using neural networks. We demonstrated through simulations that, the technique provides good control adaptability and performance. It is also noted that most of the neural network-based congestion control schemes proposed in the literature do not consider the effect of control loop delays in the development of the control algorithms because the models become too complex and intractable. In spite of the complexity introduced in our analysis by the control loop delay, the aim of the thesis is to investigate the feasibility of using neural predictive techniques to solve congestion control problems.

While we believe that our work has several claims to success, there are some weaknesses as well. Our theoretical models, though adequate for our purposes, dwell on inferences from conventional adaptive control methods and on some heuristics. We believe that even though better results can be achieved by using more sophisticated models for the analysis, this can be a daunting task since we are dealing with nonlinear systems which are not easily amenable to the existing techniques for system analysis. Thus, this thesis is only a small step in providing congestion control solutions based on system sensitivities and adaptive control. What is heartening is that, even with the naive models used here, much can still be achieved.

## 6.2. Extensions for Further Research

We have described in this thesis dynamic resource management mechanisms using adaptive neural control techniques. The studies have given us an understanding of the capabilities and limitations of the proposed control techniques. Since the control schemes rely on good neural network architectures and training algorithms to compute the control signals to the data sources, studies may be needed to determine neural architectures and training algorithms that can adapt well to the dynamic conditions in the data transfer process. Simple cost functions were used in this thesis. Extending the schemes for other cost functions that include additional information about the control and network variables, and conditions is a possible direction for research.

In the control problems studied in Chapter 5, we found that the control algorithm produced good and acceptable solutions. However, some areas for further investigation are: 1). how far can the neural predictions go before there are significant prediction errors, 2). control models using other cost functions, and 3). investigate other neural network architectures and training schemes apart from the backpropagation algorithm.

While our focus is on congestion control in this thesis, there are other areas to which the ideas proposed in this thesis can be applied. Specifically, the presented framework on sensitivity analysis is not limited to congestion control and is also more generally applicable to other systems and problems. The sensitivity techniques described here can be applied to distributed load balancing and routing problems.

In most distributed computing systems, the work generated at a node is processed there; little sharing of computational resources is provided. In such systems it is possible for some nodes to be heavily loaded while others are lightly loaded, resulting in poor overall system performance. The purpose of load sharing is to improve performance by redistributing the workload among the nodes. One of the primary motivating factors behind the development of distributed systems has been the economic benefits which can be realized by sharing expensive resources among computational agents within a distributed environment. These shared resources may be either software (e.g., a software module or a database file) or hardware (e.g., a printer, memory, or CPU). A fundamental question which arises in such systems is how to share these resources optimally among the distributed network agents. The load sharing policies with the greatest potential benefit are adaptive in the sense that they react to changes in the system state. Adaptive policies can range from simple to complex in their acquisition and use of system state information. Research in this area can address the problem of obtaining on-line sensitivity information using the techniques described in this thesis, and integrating it into a distributed resource sharing algorithm as suggested in [Cass88a][Cass88b][Kuro86]. These algorithms require performance sensitivity information with respect to the load balancing policy parameters, which is generally hard to obtain analytically (either in closed-form or numerically) for systems of practical interest without imposing restrictive assumptions on the system model.

The routing of packets from source node to destination node is another important issue in the design of packet-switched networks, since it affects several performance measures of interest. By routing we mean the set of decisions regarding the outgoing link to be used for transmitting messages at each network node. The objective of a routing algorithm is to

optimize some performance measure, e.g., the mean packet delay or network throughput. Routing may be done in a centralized or distributed manner. In the former case, a special node in the network, called the routing control center (RCC), periodically receives information from all other network nodes and, based on this global information, it sets up and updates routing tables for all nodes. This method suffers from high communication overhead, and must deal with the problem of handling link and node failures. An even more serious problem is how to handle the failure of the RCC itself. Distributed routing avoids some of these problems. In this case, each node makes its own routing decision based on the (local) information it receives from its neighboring nodes. If this approach is to be used, one potential problem here which must be addressed is that inconsistent routing paths can cause looping of packets and possibly deadlocks.

In general, the optimal routing problem reduces to minimizing (or maximizing) some objective function $D(f_{ik})$ with respect to the variables $f_{ik}$, where $f_{ik}$ denotes the flow on link $(i, k)$, subject to certain constraints such as flow conservation and nonnegativity of flows. Optimal routing algorithms may differ in the precise formulation of the problem, and in the choice of the objective function. However, a common feature is the requirement for knowledge of the derivatives of the performance measure (e.g., the mean packet delay) with respect to the control (e.g., the link flows $f_{ik}$). Therefore, it becomes important to be able to estimate these derivatives as efficiently and accurately as possible. Gallager [Gall77], for example, has defined an interesting algorithm to solve the distributed routing problem in a quasistatic environment. A quasistatic environment is one where the offered traffic statistics for each source-destination pair change slowly over time, and individual

traffic functions do not show large and persistent deviations from the average. Like most other optimal routing algorithms (e.g., [Bert84][LeeK87][Tsit84]), Gallager's algorithm assumes knowledge of the delay gradient $D_{ik}(f_{ik})$. The question is: how does one determine the derivatives of delays over the links? It can be done analytically if an appropriate formula giving the link delay as a function of the link flows is available. For example, Kleinrock [Klei64] has used the following formula for the average total delay(s) over a link, based on the assumption that the link behaves like an M/M/1 queueing system:

$$D_{ik}(f_{ik}) = \frac{f_{ik}}{C_{ik} - f_{ik}},$$

where $f_{ik}$ denotes the amount of traffic on link $(i, k)$, and $C_{ik}$ is the capacity of $(i, k)$. This expression, however, is based on the assumptions of Poisson arrivals at nodes, exponentially distributed packet lengths, and the "independence assumption" of service times at successive nodes (i.e., when a packet arrives at a node, a new length is assigned to it from some common exponential distribution). Such assumptions do not hold in practice. Hence, instead of seeking closed-form expressions for $D_{ik}(f_{ik})$, it is preferable to estimate the derivative of the delay (i.e., the sensitivities) directly, by using the techniques described in this thesis (particularly Chapter 2) and data available in a real network.

In conclusion, we feel that the area of congestion control is vast, and still growing. We hope that this thesis makes a contribution to the field.

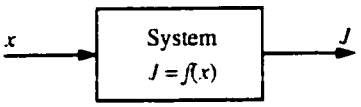
# Appendix A: Statistical System Variability Analysis using Sensitivities

## A.1. Formulation of Statistical Analysis Problem

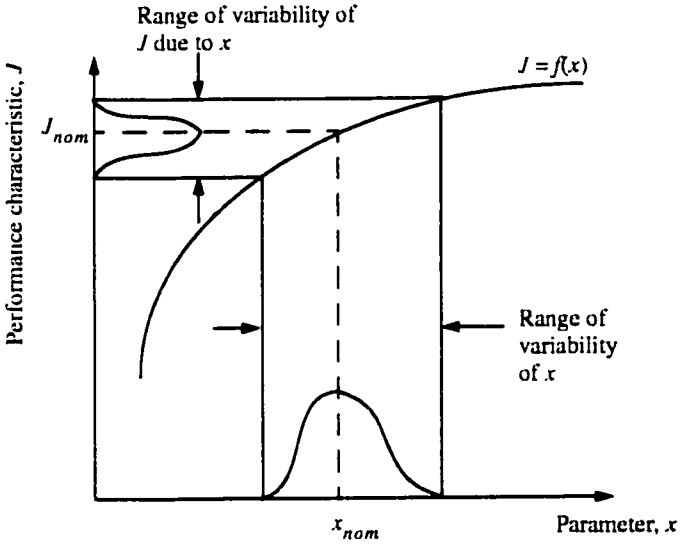
The performance of a system depends on the parameters of the system and the particular set of values assigned to those parameters. The design of a system is based on the nominal parameter values, but the actual values can be distributed around the design values due to system variations. Therefore the statistical analysis to find the statistical distribution of the system performance (or operating characteristics) corresponding to an actual distribution of the system parameters is important especially from the viewpoint of reliability improvement. Here, we discuss how first order sensitivities can be used for characterizing the statistical distribution [Mark65]. Discussion on the application of second-order sensitivities for characterizing the statistical distribution can be found in [Aiha73] and [Cram66].

The dependence of the system performance on the parameters of the system and the particular set of values assigned to those parameters can be exploited by a system designer or a control mechanism to change parameter values to obtain the desired performance. The concept of system variability is illustrated in Figure A1 where a performance characteristic $J$ of a system is plotted as a function of parameter $x$. The effect on $J$ of this variability in $x$ can be determined by projecting the $x$ distribution up to the curve $J = f(x)$ and over to the $J$ axis as shown. It is evident that, if the curve is essentially linear, the distribution of $J$ will have basically the same shape as the distribution of $x$. Similarly, if the curve is highly

nonlinear in the range of interest, the distribution of $J$ will be a distorted version of that of $x$. The inevitable distribution of the system parameter values demand that performance tolerance bounds be associated with the parameters. The tolerance bounds define a range of possible values that a system parameter can have.




a). System




b). Performance

**FIGURE A1. Performance variability of a system.**

Denote the nominal input parameters by the following vector

$$\bar{x}_0 = [x_1, x_2, \ldots, x_N]^T. \tag{A.1}$$

If the nominal parameter values are given by (A.1), variations in the input parameters and the combined effect of $N$ parameters variations $\Delta x_i$, $i = 1, \ldots, N$, on the system performance $J$, result in the change

$$\Delta J = \sum_{i=1}^{N} \frac{\partial J}{\partial x_i} \Delta x_i + \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\partial J}{\partial x_i \partial x_j} \Delta x_i \Delta x_j + \ldots \tag{A.2}$$

in which higher-order derivatives of the Taylor-series expansion for $\Delta J$ are neglected. There are two ways of quantifying the concept of input parameter variations:

1. By specifying the minimum and maximum values $x_m$ and $x_M$ associated with each nominal parameter $x_0$, where $x_m \leq x_0 \leq x_M$.

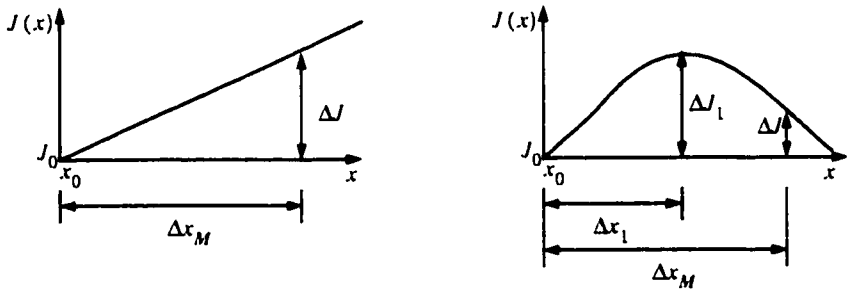
2. By assuming a certain probability density function for the system parameters and specifying the mean value $\mu$ and the standard deviation $\sigma$ (or variance $\sigma^2$).

The characteristic variations of $N$ parameters are represented by the vector $\bar{w}_0 = [w_1, w_2, \ldots, w_N]^T$. Each value $w_i$, $i = 1, \ldots, N$ may have the meaning of either maximum variation (i.e., $\Delta x_m = x_0 - x_m$ or $\Delta x_M = x_M - x_0$) or the standard deviation $\sigma_i$.

## A.2. System Response Computation based on Input Parameter Limits

By selecting the proper minimum or maximum values of the system input parameters $x$, the upper and lower limits of the output function of the system can be evaluated. Differen-

tial sensitivities (derivatives) can be used in this case. Suppose the system sensitivities are available in the form of partial derivatives of the system response $J$ with respect to parameters $x$. Then the system response at the extreme values $x + \Delta x_M$ can be easily computed, provided that a linear relation between the system response and the parameter changes can be assumed (see Figure A2a). The computation of $\Delta J$ is based on (A.2), in which second- and higher-order terms are neglected. In Figure A2b, a nonlinear system function $J(x)$ is plotted against a system parameter $x$. The first-order computation using the formula $\Delta J = (\partial J/\partial x)\,\Delta x_M$, which is permitted in Figure A2a, is not allowed in Figure A2b. In the latter case, a parameter change $\Delta x_M$ causes a smaller deviation of $J(x)$ than $\Delta x_1$, which is smaller than $\Delta x_M$. The reason is that the derivative $dJ/dx$ changes sign in the input parameter range. This example illustrates the influence of nonlinear effects on the computation of $\Delta J$ when large input parameter variations are included in the computation.




Computation of $\Delta J$ at a parameter change $\Delta x_M$

**FIGURE A2. Example system response due to input parameter variations.**

## A.3. Statistical Method for System Response Computation

Another method for system response analysis is based on a direct computation of the variance of the system response, when the variances of the system input parameters are given. Assuming that small parameter variations have a normal (Gaussian) distribution, system response can easily be computed. This method is referred to as the moment method [Mark65]. Note that the variance $\sigma^2$ of a random variable $x$ can be identified with the second moment. The moment method allows us to derive the distribution of an output response of a system when the distribution of the system parameters are given. Two assumptions are implied in the moment method:

1. The probability density functions of the system parameters are Gaussian. When the actual distribution are not Gaussian, they are assumed to be Gaussian with the same mean value and variance.

2. There is a linear relationship between a parameter variation $\Delta x_i$ and the output response variation $\Delta J$. In fact, we assume the first-order approximation of (A.2),

$$\Delta J = \sum_{i=1}^{N} \frac{\partial J}{\partial x_i} \Delta x_i, \qquad i = 1, \ldots, N. \tag{A.3}$$

If the variance $\sigma_i^2$ (and mean $\mu_i$) is associated with the variable $x_i$, the resulting effect on the variance of the output response $J$ will be $\sigma_{Ji} = (\partial J/\partial x_i)\,\sigma_i$ and the total variance of $J$ is given by

$$\sigma_J^2 = \sum_{i=1}^{N} \left(\frac{\partial J}{\partial x_i}\right)^2 \sigma_i^2, \quad i = 1, \ldots, N \tag{A.4}$$

and if the effect of correlation is taken into consideration, then we have

$$\sigma_J^2 = \left[\frac{\partial J}{\partial x_1}\sigma_1 \cdots \frac{\partial J}{\partial x_N}\sigma_N\right] \bar{R} \left[\frac{\partial J}{\partial x_1}\sigma_1 \cdots \frac{\partial J}{\partial x_N}\sigma_N\right]^T. \tag{A.5}$$

The variable $\bar{R}$ is the correlation matrix for $N$ random variables and is defined as

$$\bar{R} = \begin{bmatrix} 1 & \rho_{12} & \cdots & \rho_{1N} \\ \rho_{21} & 1 & \cdots & \rho_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ \rho_{N1} & \rho_{N2} & \cdots & 1 \end{bmatrix}$$

where $\rho_{ij} \in [-1, 1]$ is the correlation coefficient of two variables $x_i$ and $x_j$, $i, j = 1, \ldots, N$ and is given by

$$\rho_{ij} = \rho_{ji} = \frac{\sum_{k=1}^{N} (x_{ik} - \mu_i)(x_{jk} - \mu_j)}{N\sigma_i\sigma_j}, \quad i, j = 1, \ldots, N.$$

The variance $v_x^2$ of the random variable $x = x_1 + x_2 + \ldots + x_N$ is given by

$$v_x^2 = \bar{v}^T \bar{R} \bar{v}$$

where $\bar{v} = [\sigma_1, \ldots, \sigma_N]^T$. When the random variables are uncorrelated, $R = 1$ and

$v_x^2 = \sigma_1^2 + \ldots + \sigma_N^2$.

Hence, the variance of the output response can be directly evaluated, if the distribution parameters of the system parameters are given. The moment method (A.5) is based on the first-order relation (A.3). The effect of correlation can also be taken into consideration by properly substituting correlation coefficients in the correlation matrix $\bar{R}$ in (A.5). By using the multivariable Taylor-series expansion of the response function, the variance of the output function can be expressed as [Mark65]

$$\sigma_J^2 = \sum_{i=1}^{N} \left(\frac{\partial J}{\partial x_i}\right)^2 \sigma_i^2 + 2 \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(\frac{\partial J}{\partial x_i}\right)\left(\frac{\partial J}{\partial x_j}\right) \sigma_i \sigma_j \rho_{ij}, \tag{A.6}$$

where $\sigma_i$ and $\sigma_j$ are the standard deviations of the parameters $x_i$ and $x_j$ respectively, $\rho_{ij}$ is the correlation coefficient, and $N$ is the number of the system parameters. Eq. (A.6) is referred to the propagation-of-variance formula [Mark65]. The propagation-of-variance formula is a mathematical statement that performance-characteristic variability is the net result of the variability of all parameters in the system, and, furthermore, that the contribution of each parameter depends upon its individual variability and the relative importance of that parameter in determining the performance characteristics.

The first term in (A.6) includes the variance of each parameter and the partial derivative of the performance characteristic (with respect to that parameter). The factors in this term are squared and, therefore, always positive. The second term in the equation includes each pair

of correlated parameters. This term can be either positive or negative, simulating the situation in an actual system where the correlation between two parameters can act to either increase or decrease the over-all performance variability.

# Appendix B: Second-Order Sensitivity Analysis

The second-order partial derivatives of the network outputs with respect to the inputs are calculated using a backward chaining rule similar to that used for the first order derivatives in Chapter 2. These partial derivatives represent the output second order sensitivities [HashS92][Tsou97]. Below, we give a complete derivation of these sensitivities.

- For the neurons in layer $N$:

$$\frac{\partial^2 y_k}{\partial h_i^N \partial h_j^N} = \frac{\partial}{\partial h_i^N}\left[\alpha y_k (1 - y_k) \cdot w_{jk}^N\right] = \alpha w_{jk}^N \cdot \frac{\partial}{\partial h_i^N}\left[y_k - y_k^2\right] \tag{B.1}$$

$$= \alpha w_{jk}^N \cdot (1 - 2y_k) \frac{\partial y_k}{\partial h_i^N} = \alpha^2 (1 - 2y_k)\, y_k (1 - y_k) \cdot w_{jk}^N \cdot w_{ik}^N,$$

$$\forall (i, j, k).$$

- For the neurons in the remaining hidden layers (layer $p$, $p = N-1, \ldots, 1$):

$$\frac{\partial^2 y_k}{\partial h_i^p \partial h_j^p} = \frac{\partial}{\partial h_i^p}\left[\sum_l \frac{\partial y_k}{\partial h_l^{p+1}} \alpha h_l^{p+1}\left(1 - h_l^{p+1}\right) w_{jl}^p\right]$$

$$= \alpha \sum_l w_{jl}^p \left[\left\{\frac{\partial}{\partial h_i^p}\left(h_l^{p+1} - \left(h_l^{p+1}\right)^2\right)\right\} \frac{\partial y_k}{\partial h_l^{p+1}} + h_l^{p+1}\left(1 - h_l^{p+1}\right) \frac{\partial}{\partial h_i^p}\left(\frac{\partial y_k}{\partial h_l^{p+1}}\right)\right]$$

Simplifying further, we have

$$\frac{\partial^2 y_k}{\partial h_i^p \partial h_j^p} = \alpha \sum_l w_{jl}^p \left[ \left( 1 - 2h_l^{p+1} \right) \frac{\partial h_l^{p+1}}{\partial h_i^p} \frac{\partial y_k}{\partial h_l^{p+1}} + h_l^{p+1} \left( 1 - h_l^{p+1} \right) \frac{\partial}{\partial h_i^p} \left( \frac{\partial y_k}{\partial h_l^{p+1}} \right) \right]$$

$$= \alpha \sum_l w_{jl}^p \left[ \left( 1 - 2h_l^{p+1} \right) \alpha h_l^{p+1} \left( 1 - h_l^{p+1} \right) w_{il}^p \frac{\partial y_k}{\partial h_l^{p+1}} + h_l^{p+1} \left( 1 - h_l^{p+1} \right) \frac{\partial}{\partial h_i^p} \left( \frac{\partial y_k}{\partial h_l^{p+1}} \right) \right], \tag{B.2}$$

where

$$\frac{\partial}{\partial h_i^p} \left( \frac{\partial y_k}{\partial h_l^{p+1}} \right) = \sum_s \left( \frac{\partial^2 y_k}{\partial h_s^{p+1} \partial h_l^{p+1}} \right) \left( \frac{\partial h_s^{p+1}}{\partial h_i^p} \right)$$

$$= \sum_s \left( \frac{\partial^2 y_k}{\partial h_s^{p+1} \partial h_l^{p+1}} \right) \left[ \alpha h_s^{p+1} \left( 1 - h_s^{p+1} \right) w_{is}^p \right]. \tag{B.3}$$

Thus,

$$\frac{\partial^2 y_k}{\partial h_i^p \partial h_j^p} = \alpha^2 \sum_l w_{jl}^p \left[ \left( 1 - 2h_l^{p+1} \right) h_l^{p+1} \left( 1 - h_l^{p+1} \right) w_{il}^p \frac{\partial y_k}{\partial h_l^{p+1}} + h_l^{p+1} \left( 1 - h_l^{p+1} \right) \sum_s \left( \frac{\partial^2 y_k}{\partial h_s^{p+1} \partial h_l^{p+1}} \right) \left[ h_s^{p+1} \left( 1 - h_s^{p+1} \right) w_{is}^p \right] \right], \tag{B.4}$$

$\forall (i, j, k)$ .

- For the input layer (layer 0):

$$\frac{\partial^2 y_k}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left[\sum_l \frac{\partial y_k}{\partial h_l^1}\alpha h_l^1\left(1-h_l^1\right)w_{jl}^0\right] \tag{B.5}$$

$$= \alpha\sum_l w_{jl}^0\left[\left(1-2h_l^1\right)\alpha h_l^1\left(1-h_l^1\right)w_{il}^0\left(\frac{\partial y_k}{\partial h_l^1}\right) + h_l^1\left(1-h_l^1\right)\left(\frac{\partial^2 y_k}{\partial x_i \partial h_l^1}\right)\right],$$

$$\forall (i,j,k)\,,$$

where

$$\frac{\partial^2 y_k}{\partial x_i \partial h_l^1} = \sum_s\left(\frac{\partial^2 y_k}{\partial h_s^1 \partial h_l^1}\right)\left[\alpha h_s^1\left(1-h_s^1\right)w_{is}^0\right]. \tag{B.6}$$

Thus,

$$\frac{\partial^2 y_k}{\partial x_i \partial x_j} = \alpha^2\sum_l w_{jl}^0\left[\left(1-2h_l^1\right)h_l^1\left(1-h_l^1\right)w_{il}^0\frac{\partial y_k}{\partial h_l^1}\right. \tag{B.7}$$

$$\left. + h_l^1\left(1-h_l^1\right)\sum_s\left(\frac{\partial^2 y_k}{\partial h_s^1 \partial h_l^1}\right)\left[h_s^1\left(1-h_s^1\right)w_{is}^0\right]\right],$$

$$\forall (i,j,k)\,.$$

Higher order sensitivities may be obtained in a similar fashion, however, for most practical applications, the first and second order sensitivities are the most important ones.

# Appendix C: A Neural Adaptive PID-Like Controller

PID (Proportional-Integral-Derivative) control is a well established and popular technique for industrial process control [Frank94]. This is due to its simplicity both from the design and parameter tuning point of view. However, the capability of PID controllers are limited to linear systems. PID controllers have difficulties in controlling complex systems that are nonlinear, especially when system dynamics have parameter and structure uncertainties. Here we present an adaptive neural PID-like controller using neural network modeling for learning the characteristics of a dynamic system [JinW97]. The purpose of the neural network is to learn the characteristics of the dynamic (nonlinear) system for the on-line solution of the system sensitivities.

Let us define the following cost function:

$$J = J_1 + J_2 + J_3 = \frac{1}{2}e^2(n+1) + \frac{1}{2}\beta e^{\,2}(n+1) + \frac{1}{2}\gamma e^{\,2}(n+1), \qquad \text{(C.1)}$$

where $e(n+1) = r(n+1) - \hat{y}(n+1)$, $r(n+1)$ is the setpoint, $\hat{y}(n+1)$ is the predicted output of the system, $e\,(n+1) = e(n+1) - e(n)$, $e\,(n+1) = e\,(n+1) - e\,(n)$ and $\beta, \gamma$ are weighting factors. Recall that the neural model of the system dynamics (described in Chapter 3) is given by

$$\hat{y}(n+1) = \hat{f}[\lambda(n), \lambda(n-1), \ldots, \lambda(n-m), y(n), y(n-1), \ldots, y(n-l); V],$$

where $V$ represents the weight matrix of the neural network.

The sensitivity $\partial J/\partial u(n)$ of the cost function $J$ with respect to $u(n)$ is obtained as

$$\frac{\partial J}{\partial u(n)} = \frac{\partial J_1}{\partial u(n)} + \frac{\partial J_2}{\partial u(n)} + \frac{\partial J_3}{\partial u(n)}$$

where

$$\begin{aligned}\frac{\partial J_1}{\partial u(n)} &= -e(n+1)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)}\frac{\partial \lambda(n)}{\partial u(n)} \\ &= -e(n+1)\psi(n)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)},\end{aligned}$$

$$\begin{aligned}\frac{\partial J_2}{\partial u(n)} &= e(n+1)\frac{\partial [e(n+1) - e(n)]}{\partial \lambda(n)}\frac{\partial \lambda(n)}{\partial u(n)} \\ &= e(n+1)\psi(n)\frac{\partial e(n+1)}{\partial \lambda(n)} \\ &= -e(n+1)\psi(n)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)},\end{aligned}$$

and

$$\begin{aligned}\frac{\partial J_3}{\partial u(n)} &= e(n+1)\frac{\partial [e(n+1) - e(n) - e(n) + e(n-1)]}{\partial \lambda(n)}\frac{\partial \lambda(n)}{\partial u(n)} \\ &= e(n+1)\psi(n)\frac{\partial e(n+1)}{\partial \lambda(n)} \\ &= -e(n+1)\psi(n)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)}.\end{aligned}$$

Using the gradient descent rule

$$u(n+1) = u(n) - \eta \frac{\partial J}{\partial u(n)}$$

and (C.1) we obtain the following expression

$$u(n+1) = u(n) + \eta e(n+1)\psi(n)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)} + \eta \beta e(n+1)\psi(n)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)}$$
$$+ \eta \gamma e(n+1)\psi(n)\frac{\partial \hat{y}(n+1)}{\partial \lambda(n)},$$

from which we obtain the following PID-like adaptive control algorithm

$$\begin{aligned} u(n+1) &= u(n) + \eta e(n+1)\psi(n)\xi + \eta \beta e(n+1)\psi(n)\xi \\ &\quad + \eta \gamma e(n+1)\psi(n)\xi \\ &= u(n) + Pe(n+1) + Ie(n+1) + De(n+1), \end{aligned} \tag{C.2}$$

where

$$\xi = \frac{\partial \hat{y}(n+1)}{\partial u(n)}; P = \eta \beta \xi \psi(n); I = \eta \xi \psi(n); D = \eta \gamma \xi \psi(n).$$

If we use the sign of $\partial \hat{y}(n+1)/\partial u(n)$ instead of $\partial \hat{y}(n+1)/\partial u(n)$ itself in (C.2), the well-known incremental form of PID algorithm is obtained. In this case, even if the sensitivity $\partial \hat{y}(n+1)/\partial u(n)$ is unknown, the accurate value of $|\partial \hat{y}(n+1)/\partial u(n)|$ is not important, because the step size can be adjusted by setting $\eta$. Certainly, this requires that $|\partial \hat{y}(n+1)/\partial u(n)| < \infty$, $\forall n$. Therefore, if the sign of $\partial \hat{y}(n+1)/\partial u(n)$ at each instant is known, then we can get a simple algorithm to control the system. Obviously, the

sign of $\partial\hat{y}(n+1)/\partial u(n)$ is easy to determine for most systems using the neural sensitivity methods discussed in Chapter 2. Thus, the control algorithm becomes

$$u(n+1) = u(n) + \eta\psi(n)\,[e(n+1) + \beta e\ (n+1) + \gamma e\ \ (n+1)]\ \mathrm{sgn}\,[\xi]\ , \qquad \text{(C.3)}$$

where $\mathrm{sgn}\,[\xi]$ denotes the sign of $\xi$.

# Appendix D: Weighted Predictive Control

## Performance Index

The performance index (or cost function) required for the predictive control technique described in Chapter 5 can also be formulated in terms of Clarke's weighted predictive control performance index [ClarW87]. The future control signal sequence is obtained by minimizing the following performance index

$$J = \frac{1}{2}\sum_{i=d}^{L}[r(n+i) - \hat{y}(n+i)]^2 + \frac{1}{2}\sum_{j=1}^{N}\gamma_j[\Delta u(n+j-1)]^2, \tag{D.1}$$

where $L$ is the predictive horizon, $N$ is the control horizon and $\gamma$ is the control weighting sequence by which jump magnitude in $u$ is suppressed (i.e., $\gamma$ is a damping factor). The control algorithm can be obtained by applying the gradient descent rule to minimizing the cost function. In each sampling period, we have

$$\Delta U^k = -\eta\frac{\partial J}{\partial U^k}, \tag{D.2}$$

where $\Delta U^k = [\Delta u^k(n), \ldots, \Delta u^k(n+N-1)]^T$, $\Delta u^k(n) = u^k(n) - u^k(n-1)$, $\eta$ is the control step size, and

$$\frac{\partial J}{\partial U^k} = -\frac{\partial \hat{Y}^k}{\partial U^k}E^k + \Gamma\Delta U^k, \tag{D.3}$$

where $\Gamma = diag\,[\gamma_1, \gamma_2, \ldots, \gamma_N]$ . In matrix form, the cost function can be expressed as

$$J = \frac{1}{2}[E^T E] + \frac{1}{2}[\Delta U^T \Gamma \Delta U] \ . \tag{D.4}$$

Thus,

$$\Delta U^k = -\eta \frac{\partial J}{\partial U^k} = \eta \frac{\partial \hat{Y}^k}{\partial U^k} E^k - \eta \Gamma \Delta U^k = (I + \eta\Gamma)^{-1} \eta \frac{\partial \hat{Y}^k}{\partial U^k} E^k, \tag{D.5}$$

where $I$ is the identity matrix. According to the principle of receding horizon control [Soet92], we have

$$u(n+1) = u(n) + [1, 0, \ldots, 0]\,(I + \eta\Gamma)^{-1} \eta \frac{\partial \hat{Y}^k}{\partial U^k} E^k. \tag{D.6}$$

If the control horizon $N = 2$, then

$$(I + \eta\Gamma)^{-1} = \frac{1}{(1+\eta\gamma_1)(1+\eta\gamma_2)} \begin{bmatrix} (1+\eta\gamma_2) & 0 \\ 0 & (1+\eta\gamma_1) \end{bmatrix}, \tag{D.7}$$

and

$$u(n+1) = u(n) + \frac{\eta}{(1+\eta\gamma_1)} \sum_{j=d}^{L} e(n+j) \frac{\partial \hat{y}(n+j)}{\partial u(n)}. \tag{D.8}$$

Inverting the matrix $(I + \eta\Gamma)$ when $N$ is large involves additional computational complexity in the control algorithm, so a choice of $N = 2$ is normally used.

# References


[Agne76]. Agnew, C., "Dynamic Modeling and Control of Congestion-prone Systems," *Operations Research*, vol. 24, no. 3, 1976, pp. 400 - 419.

[Aiha73]. Aihara, Ken-ichi, and H. Hirayama, "Statistical Network Analysis using First- and Second-Order Sensitivities," *Electronics and Commun. in Japan*, vol. 56-A, no. 2, 1973, pp. 14 - 22.

[Arul96]. Arulambalam, A., X. Chen and N. Ansari, "Allocating Fair Rates for Available Bit Rate Service in ATM Networks," *IEEE Commun. Mag.*, Nov. 1996, pp. 92 - 100.

[ATMF96]. ATM Forum, *Traffic Management Specification*, Version 4.0, April 1996.

[Awey98a]. Aweya, J., Q. J. Zhang and D. Y. Montuno, "A Direct Adaptive Neural Controller for Flow Control in Computer Networks," *IEEE Int. Joint Conf. on Neural Networks* (*IJCNN'98*), Anchorage, Alaska, May 5-9, 1998, pp. 140 - 145.

[Awey98b]. Aweya, J., Q. J. Zhang and D. Y. Montuno, "A Neural Network-based Congestion Control Scheme with Fuzzy Parameter Adaptation for Computer Networks," *IEEE Int. Conf. on Fuzzy Systems* (*IEEE-FUZZ'98*), Anchorage, Alaska, May 5-9, 1998.

[Awey98c]. Aweya, J., Q.J. Zhang and D. Y. Montuno, "Modeling and Control of Dynamic Queues in Computer Networks using Neural Networks," *Int. Conf.*

*on Intelligent Systems and Control (ISC'98)*, Halifax, Canada, June 1-4, 1998, pp. 144 - 151.

[Awey98d]. Aweya, J., Q.J. Zhang and D. Y. Montuno, "Neural Sensitivity Methods for the Optimization of Queueing Systems," (Invited Paper), *World Multiconference on Systemics, Cybernetics and Informatics (SCI'98)*, July 12 - 16, 1998, Orlando, Florida, pp. 638 - 645.

[Awey98e]. Aweya, J., and L. O. Barbosa, "Neurocontroller for Buffer Overload Control in a Packet Switch", *IEE Proceedings - Communications*, Vol. 145, No. 4, 1998, pp. 227 - 233.

[Awey98f]. Aweya, J., D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "A Gradient-based Binary Feedback Scheme for Congestion Control in Computer Networks," *Int. Conf. on Networks and Communication Systems (NCS'98)*, Pittsburgh, PA, May 14-16, 1998, pp. 85 - 92.

[Awey98g]. Aweya, J., D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "A Binary Feedback Flow Control Scheme using Systems Sensitivity Derivatives for Congestion Detection," *6th Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'98)*, Montreal, Canada, July 19 - 24, 1998, pp. 112 - 117.

[Awey98h]. Aweya, J., D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "Optimal Congestion Bit Setting in a Flow Control Scheme using Neural Networks," *Proc.*

*GLOBECOM'98*, Sydney, Australia, November 8 - 12, 1998, pp. 2675 - 2682.

[Awey98i]. Aweya, J., D. Y. Montuno, Q. J. Zhang and L. O. Barbosa, "Congestion Control using a Multi-Step Neural Predictive Technique," *Proc. GLOBECOM'98*, Sydney, Australia, November 8 - 12, 1998, pp. 1705 - 1714.

[Benm93]. Benmohamed, L., and S. M. Meerkov, "Feedback Control of Congestion in Packet Switching Networks: The Case of a Single Congested Node," *IEEE/ ACM Trans. Networking*, vol. 1, no. 6, Dec. 1993, pp. 693 -708.

[Benn94]. Bonnet, J., and G. T. Des Jardins, "Comments on the July PRCA Rate Control Baseline," *ATM Forum Contribution AF-TM 94-0682*, July 1994.

[Bert84]. Bertsekas, D. P., E. M. Gafni, and R. G. Gallager, "Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks," *IEEE Trans. Commun.*, vol. COM-32, 1984, pp. 911 - 919.

[Bert87]. Bertsekas, D. and R. Gallager, *Data Networks*, Englewood Cliffs, NJ: Prentice-Hall, 1987.

[Bolo90]. Bolot, J.-C., and U. Shankar, "Dynamic Behavior of Rate-based Flow Control Mechanisms," *ACM SIGCOMM, Comp. Commun. Rev.*, vol. 30, 1990, pp. 35 - 49.

[Bray80]. Brayton, R. K. and R. Spence, *Sensitivity and Optimization*, Elsevier, Amsterdam, 1980.

[Carr70]. Carr, S., S. Crocker and V. Cerf, "HOST-HOST Communication Protocol in the ARPA Network," *AFIPS Conference Proceedings, Spring Joint Computer Conference*, Atlantic City, NJ, May 1970, vol. 36, pp. 589 - 597.

[Cass88a]. Cassandras, C. G., and J. I. Lee, "Applications of Perturbation Techniques to Optimal Resource Sharing in Discrete Event Systems," *Proc. 1988 American Control Conf.*, 1988, pp. 450 - 455.

[Cass88b]. Cassandras, C. G., and J. I. Lee, "Discrete Events Systems with Real-Time Constraints: A Distributed Algorithm for Optimal Resource Sharing," *Proc. 27th Conf. Decision and Control*, Austin, Texas, Dec. 1988, pp. 1508 - 1513.

[Cass93]. Cassandras, C. G., *Discrete Event Systems: Modeling and Performance Analysis*, Aksen Associates, Inc. 1993.

[Cerf74]. Cerf, V. G., and R. E. Kahn, "A Protocol for Packet Network Interconnection," *IEEE Trans. Commun.*, vol. COM-22, no. 5, May 1974, pp. 637 - 648.

[ChenX92]. Chen, X., and I. M. Leslie, "Neural Adaptive Congestion Control for Broadband ATM Networks," *IEE Proc.-I*, vol. 139, no. 3, Jun. 1992, pp. 233 - 240.

[ChenT96]. Chen, T. M., S. S. Liu and V. K. Samalam, "The Available Bit Rate Service for Data in ATM Networks," *IEEE Commun. Mag.*, May 1996, pp. 56 - 71.

[Cher89]. Cherton, D., "Sirpent: A High-Performance Internetworking Approach," *Proc. ACM SigComm 1989*, Sept. 1989, pp. 158 - 169.

[Chiu89]. Chui, D. and R. Jain, "Analysis of Increase and Decrease Algorithms for Congestion Avoidance in Computer Network," *Computer Networks and ISDN Systems*, vol. 17, no. 1, 1989, pp. 1 - 14.

[ClarD87]. Clark, D. D., M. L. Lambert and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol," *RFC 998*, Network Working Group, March 1987.

[ClarW87]. Clarke, D. W., C. Mohtadi, and P. S. Tuffs, "Generalized Predictive Control," Part I and Part II, *Automatica*, vol. 23, 1987, pp. 137 - 160.

[Cook91]. Cook, D., and R. Shannon, "A Sensitivity Analysis of a Backpropagation Neural Network for Manufacturing Process Parameters," *Journal of Intelligent Manufacturing*, vol. 2, pp. 155 - 163, 1991.

[Cram66]. Cramer, H., *Mathematical Methods of Statistics*, Princeton Univ. Press, 1966.

[Cruz73]. Cruz, J. B., (ed.), *System Sensitivity Analysis*, Dowden, Hutchinson and Ross, Stroudsberg, 1973.

[Cybe89]. Cybenko, G., "Approximation by Superposition of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, Springer-Verlag, New York, vol. 2, pp. 303 - 314, 1989.

[Davi72]. Davies, D. W., "The Control of Congestion in Packet Switching Networks," *IEEE Trans. Commun.* vol. 20, June 1972, pp. 546 - 550.

[Duda83]. Duda, A., "Transient Diffusion Approximations for Some Queueing Systems," *INFOR*, pp. 118 - 128, 1983.

[Ekbe89]. Ekberg, A. E., D. T. Luan and D. M. Lucantoni, "Bandwidth Management: A Congestion Control Strategy for Broadband Packet Networks: Characterizing the Throughput-Burstness Filter," *Proc. ITC Specialist Seminar*, Adelaide, 1989.

[Fili88]. Filipiak, J., *Modeling and Control of Dynamic Flows in Communication Networks*, Springer-Verlag, 1988.

[Floy91]. Floyd, S. and V. Jacobson, "Traffic Phase Effects in Packet-Switched Gateways," *Computer Commun. Rev.*, vol. 21, no. 2, April 1991.

[Fran70]. Frank, M., I. T. Frisch and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," *AFIPS Conference Proceedings, Spring Joint Computer Conference*, Atlantic City, NJ, May 1970, vol. 36, pp. 581 - 587.

[Frank94]. Franklin, G. F., J. D. Powell and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Addison-Wesley, 1994.

[Funa89]. Funahashi, K., "On the Approximate Realization of Continuous Mappings by Neural Networks," *Neural Networks*, vol. 2, pp. 183 - 192, 1989.

[Gall77]. Gallager, R. G., "A Minimum Delay Routing Algorithm using Distributed Computation," *IEEE Trans. Commun.*, vol. COM-25, Jan. 1977, pp. 73 - 85.

[Gerl80]. Gerla, M. and L. Kleinrock, "Flow Control: A Comparative Study," *IEEE Trans. Commun.*, vol. COM-28, vol. 4, April 1980, pp. 553 - 574.

[Gros85]. Gross, D., and C. M. Harris, *Fundamentals of Queueing Theory*, 2nd ed., John Wiley & Sons, New York, 1985.

[Guo92]. Guo, Z., and R. E. Uhrig, "Use of Artificial Neural Networks to Analyze Nuclear Power Plant Performance," *Nuclear Technology*, vol. 99, July 1992, pp. 36 - 42.

[Habi96]. Habib, I. W., "Applications of Neurocomputing in Traffic Management of ATM Networks," *Proc. of the IEEE*, vol. 84, no. 10, Oct. 1996, pp. 1430 - 1441.

[HashE89]. Hashem, E., "Analysis of Random Drop for Gateway Congestion Control," *LCS-Tech. Rep. 465*, Lab. for Computer Sci., Massachussetts Inst. of Tech., Cambridge, MA, 1989.

[HashS92]. Hashem, S., "Sensitivity Analysis for Feedforward Artificial Neural Networks with Differentiable Activation Functions," *Proc. IJCNN*, Baltimore, MD, June 1992, pp. I-419 - I-424.

[Hear70]. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *AFIPS Conference Proceedings, Spring Joint Computer Conference*, Atlantic City, NJ, May 1970, vol. 36, pp. 551 - 567.

[Hech89]. Hecht-Nielson, R., "Theory of the backpropagation neural network," *Proc. 1989 Int'l Joint Conf. Neural Networks*, pp. I: 593 - 605, 1989.

[Horn89]. Hornik, K., M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, vol. 2, pp. 359 - 368, 1989.

[Horn90]. Hornik, K., M. Stinchcombe, and H. White, "Universal Approximation of an Unknown Mapping and its Derivatives using Multilayer Feedforward Networks," *Neural Networks*, vol. 3, pp. 551 - 560, 1990.

[IEEE95]. *IEEE Communications Magazine*, Oct. 1995, Issue on Neurocomputing in High-Speed Networks.

[IEEE97]. *IEEE Jour. Sel. Areas Comm.*, vol. 15, no. 2, Sept. 1997, Special Issue on Neural Networks.

[IETF94]. *IETF RFC 1633*, "Integrated Services in the Internet Architecture: An Overview," R. Braden, D. Clark, S. Shenker, June 1994.

[Irla75]. Irland, M., "Simulation of CIGALE 1974," *Proc. ACM-IEEE 4th Data Commun. Symp.*, P.Q., Canada, Oct. 1975.

[JacoV88]. Jacobson, V., "Congestion Avoidance and Control," *Proc. ACM SigComm*, August 1988, pp. 314 - 329.

[JacoR88]. Jacobs, R. A., "Increased Rates of Convergence Through Learning Rate Adaptation," *Neural Networks*, vol. 1, 1988, pp. 295 - 307.

[Jain89]. Jain, R., "A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *Computer Commmun. Rev.*, Oct. 1989, pp. 56 - 71.

[Jain90]. Jain, R., "Congestion Control in Computer Networks: Issues and Trends," *IEEE Network Mag.*, vol. 4, no. 3, May 1990, pp. 24 - 30.

[Jain96]. Jain, R., "Congestion Control and Traffic Management in ATM Networks: Recent Advances and a Survey," *Computer Networks and ISDN Systems*, Nov. 1996.

[JinW97]. Jin, W., G. Wenzhong, and W. Fuli, "A PID-like Controller for Nonlinear Systems," *Proc. American Control Conf.*, Albuquerque, New Mexico, June 1997, pp. 1558 - 1562.

[Kahn71]. Kahn, R. E., and W. R. Crowther, "Flow Control in a Resource-Sharing Computer Network," *Proc. 2nd ACM-IEEE Symp. on Problems of the Optimization of Data Communication*, Palo Alto, Calif., Oct. 1971, pp. 108 - 116.

[Kamo81]. Kamoun, F., "A Drop and Throttle Flow Control Policy for Computer Networks," *IEEE Trans. Commun.*, vol. COM-29, no. 4, April 1981, pp. 444 - 452.

[Kesh91a]. Keshav, S., Congestion Control in Computer Networks, *Ph.D. Thesis*, University Of California, 1991.

[Kesh91b]. Keshav, S., "A Control-theoretic Approach to Flow Control," *ACM Comp. Comm. Review*, vol. 21, no. 4, 1991, pp. 3 - 15.

[Klei64]. Kleinrock, L., *Communication Nets: Stochastic Message Flow and Delay*, New York: McGraw-Hill, 1964.

[Klei75]. Kleinrock, L., *Queueing Systems, Volumes* 1 *and* 2, John Wiley & Sons, New York, 1975 (vol. 1), 1976 (vol. 2).

[Klim91]. Klimasauskas, C., "Neural Nets Tell Why," *Dr. Dobb's Journal*, pp. 16 - 24, April 1991.

[Ko91]. Ko, K., P. P. Mishra and S. K. Tripathi, "Predictive Congestion Control in High-Speed Wide-Area Networks," P*rotocols for High Speed Networks II*, Elservier Science Publishers, North Holland, April 1991.

[Kung95]. Kung, H. and R. Morris, "Credit-Based Flow Control for ATM Networks," *IEEE Network Mag.*, vol. 9, Mar./Apr. 1995, pp. 40 - 48.

[Kuro86]. Kurose, J. F., and S. Singh, "A Distributed Algorithm for Optimum Static Load Balancing in Distributed Computer Systems," *Proc. INFOCOM'86*, 1986, pp. 458 - 468.

[LeeK87]. Lee, K. J., D. F. Towsley, and M. Choi, "Distributed Algorithms for Minimum Delay Routing with Constraints in Communication Networks," *Proc. INFOCOM'87*, 1987, pp. 188 - 201.

[Liao93]. Liao K. Q., and L. Mason, "Self-Optimizing Window Flow Control in High-Speed Data Networks," *Computer Communications*, vol. 16, no. 11, Nov. 1993, pp. 706 - 716.

[Maji79]. Majithia, J. C., M. Irland, J. L. Grange, N. Cohen and C. O'Donnell, "Experiments in Congestion Control Techniques," *Proc. Int'l. Symp. Flow Control Computer Networks*, Versailles, France, Feb. 1979, pp. 211 - 234.

[Mank90]. Mankin, A., "Random Drop Congestion Control," *Proc. ACM SigComm 1990*, Sept. 1990.

[Mark65]. Mark, D. G. and L. H. Stember, Jr., "Variability Analysis," *Electro-Technology*, July 1965, pp. 35 - 48.

[More97]. Moret, Y., and S. Fdida, "ERAQLES an Efficient Rate Algorithm for ABR," *GLOBECOM'97*, 3-8 Nov. 1997, Phoenix Arizona.

[Mura90]. Murakami, G. J., R. H. Campbell and M. Fairnan, "Pulsar: Non-Blocking Packet Switching with Shift-register Rings," Proc. ACM SigComm 1990, Sept. 1990, pp. 145 - 155.

[Nagl87]. Nagle, J., "On Packet Switches with Infinite Storage," *IEEE Trans. Commun.*, vol. COM-35, 1987, pp. 435 - 438.

[Nare90]. Narendra, K. S., and K. Parthasarathy, "Identification and Control of Dynamical Systems using Neural Networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4 - 27, 1990.

[Post81]. Postel, J., "Transmission Control Protocol," *RFC 793*, USC Information Science Institute, 1981.

[Pouz76]. Pouzin, L., "Flow Control in Data Networks - Methods and Tools," *Proc. Int'l Conf. Comp. Commun.*, Toronto, Aug. 1976, pp. 467 - 474.

[Pouz81]. Pouzin, L., "Methods, Tools and Observations on Flow Control in Packet-Switched Data Networks," *IEEE Trans. Commun.*, vol. COM-29, no. 4, April 1981, pp. 413 - 426.

[Prue87]. Prue, W. and J. Postel, "Something a Host Could Do with Source Quench: The Source QUench Introduced Delay (SQUID), *Request for Comments 1016*, Network Working Group, July 1987.

[Rada66]. Radanovic, L., (ed.), Sensitivity Methods in Control Theory, *Proc. IFAC Symp.*, Dubrovnik, 1964, Pergamon, Oxford, 1966.

[Rama88]. Ramakrishnan, K. K. and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer," *Proc. ACM SigComm*, Stanford, CA, Aug. 1988, pp. 303 - 313.

[Rama90]. Ramakrishnan, K. K. and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM Trans. Comp. Sys.*, vol. 8, no. 2, May 1990, pp. 158 - 181.

[Robe70]. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Conference Proceedings, Spring Joint Computer Conference*, Atlantic City, NJ, May 1970, vol. 36, pp. 543 - 549.

[RobeL94]. Roberts, L., "Enhanced PRCA (Proportional Rate-Control Algorithm)," *ATM Forum Contribution 94-0735R1*, Aug. 1994.

[Rume88]. Rumelhart, D. E., G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing*, vol. 1, Chap. 8, MIT Press, Cambridge, MA, 1988.

[Sabn89]. Sabnani, K. K., and A. N. Netravali, "A High Speed Transport Protocol for Datagram/Virtual Circuit Networks," *Proc. ACM SigComm 1989*, Sept. 1989, pp. 146 - 157.

[Sage68]. Sage, A. P., *Optimal System Control*, Prentice Hall, NJ, 1968.

[Schw87]. Schwartz, M., *Telecommunication Networks: Protocols, Modeling and Analysis*, Reading, MA: Addison-Wesley, 1987.

[Sidi89]. Sidi, M., W. Liu, I. Cidon and I. Gopal, "Congestion Control Through Input Rate Regulation," *Proc. Globecom'89*, Dec. 1989, pp. 1764 - 1768.

[Soet92]. Soeterboek, R., *Predictive Control: A Unified Approach*, Prentice Hall, 1992.

[SiuK96]. Siu, K., and H. Tzeng, "Intelligent Congestion Control for ABR Service in ATM Networks," *ACM SIGCOMM, Comp. Commun. Rev.*, Oct. 1996, pp. 81 - 106.

[Tane81]. Tanenbaum, A. S., *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Tarr96]. Tarraf, A., I. Habib and T. Saadawi, "A Neurocomputing Approach to Congestion Control in an ATM Multiplexer," *Journal of High Speed Networks*, vol. 5, 1996, pp. 329 - 346.

[Tomo64]. Tomovic, R., *Sensitivity Analysis of Dynamic Systems*, McGraw-Hill, New York, 1964.

[Tsit84]. Tsitsiklis, J. N., and D. P. Bertsekas, "Distributed Asynchronous Optimal Routing in Data Networks," *Proc. 23rd Conf. Decision and Control*, Las Vegas, NV, Dec. 1984, pp. 1133 - 1138.

[Tsou97]. Tsoukalas, L. H., and R. E. Uhrig, *Fuzzy and Neural Approaches in Engineering*, John Wiley & Sons, New York, 1997.

[Turn86]. Turner, J., "New Directions in Communications (or Which Way to the Information Age?)," *IEEE Commun. Mag.*, vol. 24, no. 10, Oct. 1986.

[Wang90]. Wang, Z. and J. Crowcroft, "A New Congestion Control Scheme: Slow Start and Search," Tech. Rep., University College, London, UK, Oct. 1990.

[Will91]. Williamson, C. L., and D. R. Cheriton, "Loss-Load Curves: Support for Rate-based Congestion Control in High-Speed Datagram Networks," *Proc. ACM SigComm 1991*, Sept. 1991.

[Yin94]. Yin, N., and M. Hluchyj, "On Closed-loop Rate Control for ATM Cell Relay Networks," *Proc. INFOCOM'94*, 1994, pp. 99 - 108.

[Zhan86]. Zhang, L., "Why TCP Timers Don't Work Well," *Proc. SigComm 1986*, 1986, pp. 397 - 405.

[Zhan89]. Zhang, L., A New Architecture for Packet Switching Network Protocols, *Ph.D. Thesis*, Massachusetts Inst. of Tech., July 1989.