Traffic Classification and Shaping in ATM Networks

by

Robert C. Lehr

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Electrical Engineering

Waterloo, Ontario, Canada, 1997

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

The integration of services is the driving force behind the design of the high speed data networks of today and tomorrow. These networks must be able to deliver a broad range of services and be capable of carrying diverse classes of traffic with very different source characteristics. In the case of data traffic, some delay is tolerable, however the loss of information is not. At the other extreme, some loss is tolerable for voice traffic, however delay is not. In the middle is broadcast quality video traffic, which is sensitive to both delay and loss.

In the case of Asynchronous Transfer Mode networks, the solution of these conflicting requirements is to negotiate a Traffic Contract at the User-Network Interface, which specifies a Quality of Service level and the characteristics of the source. These characteristics are used by Call Admission Control and Usage Parameter Control to protect existing connections.

Unfortunately, the determination of source characteristics by either the user or network provider is difficult, or impossible in some situations. The usual statistical methods of identifying traffic sources do not scale well to high speed networks, nor are they applicable to all traffic types. In addition, they cannot be used to identify the traffic streams emerging from applications not envisioned when these identification techniques were developed. Thus, there is a need for a method that can accurately provide a description of traffic streams in a timely manner. Three contributions are presented in order a satisfy these needs.

The proposed traffic primitive classifier can be used to classify unknown traffic streams. This is accomplished by defining simple, deterministic characteristics of

traffic streams which are collectively called traffic primitives These traffic primitives are used to define training vectors in order for a neural network to learn the classification problem. The traffic classification results show that the neural networks not only can classify deterministic sources from which they are trained, but also they can classify a wide range of random sources, such as the class of on-off sources. With the additional functionality of Traffic Primitive Histogram Identification and Stream Transition Tracking, the primitive classifier can be applied to characterize sources which are not on-off in nature. As well, the primitive classifier can be integrated into a policer to perform more complex policing actions, and to monitor traffic streams for a given set of occurrences.

In addition to the traffic primitive classifier, two additional contributions come in the form of two traffic shapers, the Minimized Variance shaper and the Burst-oriented shaper. Both shapers have the ability to produce near deterministic streams, given appropriate sources are shaped, at fairly low costs in delay and buffer size at the shapers. In the case of the Minimized Variance shaper, source information is utilized in order to find an optimal shaping parameter that has the effect of minimizing the interdeparture time variance of the stream exiting the shaper. For the case of the Burst-oriented shaper, source information is not required since it assumes that bursts and silences emerge from the ATM Adaptation Layer, and so it attempts to spread a burst into the immediately following silence period. By doing so, it has the ability to define an unshaping parameter, which when embedded into the traffic stream, can be used to unshape the source at the destination User-Network Interface. This has the dual benefits of offering the network provider

an ability to characterize sources and hence improve network efficiency, and also to allow users to treat the network as a transparent connection.

Acknowledgements

Firstly, I would like to express my thanks and gratitude to my supervisor, Professor Jon W. Mark, for his advice, guidance, encouragement, patience, effort and support over the years of my doctoral research. He always has and continues to be there for me, and I am honored to have been his student.

I would also like to thank my thesis committee, Dr. L. Deng, Dr. C. Douligeris, Dr. G. Agnew and Dr. J.W. Wong, for their suggestions to improve and support of my research.

In addition, I would like to thank Dr. I. Blake and Dr. S. Chaudhuri, and again Dr. J.W. Wong for their support and guidance over the years. They have made me feel more of a friend and colleague than a student, and for this I am grateful.

Secondly, I would like to thank all my fellow graduate students over the years, whose friendship has made my time at Waterloo both enjoyable and rewarding. Not only have I been exposed to new concepts and ideas technically, but also culturally. I would like to thank: Bill Anderson, Majid Barazande-Pour, Claude Bergeron, Xuehong Chen, Michael Cheung, Sai Chu, Dave Evans, Verna Friesen, Serag Gadelrab, Nasir Ghani, José Gonzalez, George Joy, Rob Lambert, B.J. Lee, Biswajit Nandy, Hien Nguyen, Bing Nieh, the late Jim and Diane Ohi, Jing-Fei Ren, Ramin Sabry, Gord Strachan, Meenaradchagan Vishnu, Vincent Wong, Guoliang Wu, and last, but certainly not least, Atsushi Yamada. There are many others, for whose omission from this list I humbly apologize.

In addition, I would be remiss if I did not mention my fellow graduate students and the staff of the Graduate Student Association and its Board of Directors, and

the students and faculty members of various university committees with which I was pleased to spend some time as a graduate student representative.

Lastly, I would like to thank the department staff, most notably Wendy Boles and Donna O'Brecht, who have always cheerfully guided a wayward graduate student through bureaucratic swamps and the quicksand of university regulations and procedures.

Dedication

To my loving parents, Andrew and Eva,

and brother Edward,

without whose unwavering support and sound upbringing

this would not have been possible,

and to my "extended family," John and Margaret Ferber,

Rosemary Ferber, Susan and David Murphy,

for the same reasons.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The integration of services, required by increasingly complicated computer and network applications, is the driving force behind the design and analysis of the present and future high speed data networks. While there is not yet a single network which carries an ubiquitous range of services, present day networks are already starting to merge, directed by the demand for new services: data is carried over voice networks via modem connections; telephony software allows voice to be carried by data networks; audio and video applications are carried over data networks; and cable providers will soon support a mechanism for data applications to operate over the cable (television) networks, starting simply with Internet access. Thus, whereas in the past it could be said that network technology was the propelling force behind network services and applications, changes are occurring such that network applications are now driving the technology. The common thread in the existing and emerging applications is the requirement for faster transmission speeds and increased bandwidth. In addition, a more flexible delivery of bandwidth is

desired, such as "bandwidth on demand." Hence the need for "smarter," more flexible high speed networks. In particular, this thesis will consider the example of Asynchronous Transfer Mode (ATM) networks, which are currently considered by many to be the solution to the existing problems of service integration. Much of this work, however, is applicable to other high speed networks with properties similar to ATM.

One of the major problems associated with the integration of services into a single high speed network, such as an ATM network, is that of cell-level congestion. Cell-level congestion arises from two important aspects of modern networking: network utility, and bandwidth efficiency. For the case of network utility, one basic premise of modern data networks is that they are transparent to the user. That is, data networks should be as easy to use as telephone networks — simply make a call. In this light, the network should appear to the user as a high speed pipe through which data flows. The destination user should see the same traffic stream exit the network as that which entered. In addition, the traffic streams of other users of the network, including *future* users, should have no affect whatsoever on a given connection. Much of the justification for studying traffic shaping is contained in the idea of congestion avoidance, and the presentation to the destination the same traffic stream that is input to the network.

For the case of bandwidth efficiency, the problem is studied from the corresponding perspective of the network service provider. From the network provider's point of view, the network must be optimized in order to maximize revenues. This is achieved, all other things being equal, by avoiding congestion. In order to ac-

complish this, information regarding the traffic streams to be admitted or already connected to the network is required. The bulk of this thesis proposes a novel method, employing neural networks, that can be used to classify unknown traffic streams.

It is not the purpose here to give an in-depth historical account of the evolution of telephony and packet networks. Nonetheless, a sense of history regarding the origins of these networks is required in order to understand their present state, and to justify their future form. With this in mind, Section 1.1 introduces and discusses the historical aspects of circuit switched and packet switched networks, and the desire for service integration, which has lead to the development of the Integrated Services Digital Network (ISDN) and Broadband-ISDN (B-ISDN), which employs ATM as its transport mode. Section 1.2 will then introduce the salient features of ATM as they pertain to this work, including statistical multiplexing and congestion, Quality of Service and the Traffic Contract, and the User-Network Interface. After these few sections of background of ATM networks, Section 1.3 brings together the discussion and presents a hurdle to be overcome: timely and accurate source information. The requirement for source information gives rise to the need for traffic classification and shaping.

With traffic classification and shaping introduced, Section 1.4 states the aim of this research: to classify unknown traffic streams using neural networks, and to use the source information gained via classification to improve network efficiency. Since some readers may be unfamiliar with neural networks, Section 1.5 gives a brief overview. Finally, Section 1.6 defines some terms that are used throughout

this thesis. The reader may wish to peruse this section first, or refer to it from time to time, if unfamiliar terminology or notation is encountered.

Chapter 2 contains much of the work performed, namely the design and implementation of traffic classification. Section 2.1 describes *traffic primitives*, which are integral to the neural network based traffic primitive classifier. Following this, Section 2.2 describes the neural network used to classify these primitives, as well as the neural network training methodology. The results of this training are given in Section 2.3. More importantly, Section 2.4 offers results of the operation of the neural network based primitive classifier when it is presented with heretofore unobserved traffic streams.

Two applications of traffic classification are given in Chapter 3, Traffic Primitive Histogram Identification in Section 3.1, which is a method of source characterization. In Section 3.2, Stream Transition Tracking can be used to monitor the behavior of a given source.

Two traffic shapers are proposed in Chapter 4, the Minimized Variance shaper in Section 4.1 and the Burst-oriented shaper in Section 4.2. The Minimized Variance shaper requires knowledge of the type of traffic on which it is operating, and thus can be considered to be another application of traffic classification. The Burst-oriented shaper is developed from insights gained in traffic classification, and thus does not depend on this knowledge. It does, however, sport the feature of being able to "unshape" traffic streams at a source's destination, by way of an *unshaper*. In both cases, the goal of shaping is to create a deterministic traffic stream.

Finally, this thesis concludes with Chapter 5, which provides a summary of the

contributions of this work, as well as pointing towards future work.

1.1 A Brief Background in Modern Networking

To consider the history of modern networking, one must start with the origins of telephony, which gained popularity in the early 1900's. However, as stated, the purpose here is not to give an in-depth review of telecommunications. Rather, since the reader is undoubtably familiar with a telephone, the only background of telephony required is to realize that telephone networks employ connection-oriented circuit switching, whereas the data networks which began to appear in the 1960's employ connectionless packet switching [BC84, BG92, Cha83, Sch87, Sta94, Tan88]. The following section gives an overview of the origin of modern data networks, with emphasis on how they lead to the integration of services and thus ISDN, which is the subject of Section 1.1.2. In turn, the integration of services demands increasing amounts of bandwidth. The answer to this is B-ISDN, of which an overview is given in Section 1.1.3.

1.1.1 Telephone and Data Networks

In the existing analog telephone network, or POTS (Plain Old Telephone Service), before information can be exchanged a connection must be established between the users. However, once a connection has been established, it is dedicated to the two users at its end points. Information transferred over this connection always takes the same route through the network. Contrast this to a data network offering datagram service, which is connectionless and employs packet switching. In this case, no connection between the sender and receiver is made prior to the transmission of the information. Instead, the information to be exchanged between sender and re-

ceiver contains the address of the recipient. In this sense, a connectionless network is similar to the postal system [Tan88]. Additionally, it is possible for messages between the same pair of users to take completely different routes through the network. Also of importance, since there is no end-to-end connection established, there is no bandwidth dedicated to a given connection. Therefore, instead of bandwidth being wasted if a connection becomes idle, as occurs in telephone networks, in data networks messages from *another* pair of users may utilize this bandwidth.[1] These distinctions between connectionless and connection-oriented, circuit switched and packet switched networks (and services) play an important role in present day high speed networks, since these networks are designed to carry very different — one could say diametric — services.

This brief history of modern networking, as it relates to Asynchronous Transfer Mode, begins in the early 1970's. This time witnessed the establishment of the public, government and private wide area networks (WANs). Some North American examples of these first data networks are ARPANET,[2] TYMNET, MILNET, USENET, CSNET and BITNET, and SNA [BG92, Gre84, Sta94, Tan88]. In addition, many standards were established by organizations such as International Standards Organization (ISO), Comité Consultatif International de Télégraphique et Téléphonique (CCITT),[3] and the Internet Engineering Task Force (IETF). ARPANET evolved into the ARPA internet [HHS83], and is commonly referred to as

---

[1]This is referred to as *statistical multiplexing*, and will be discussed in Section 1.2.1.

[2]These acronyms are explained in the references, however the exact names of these data networks are not pertinent to this discussion.

[3]This standards organization is a part of the International Telecommunication Union (ITU), and has recently been renamed ITU-T.

"the Internet" today. The importance of mentioning these networks is that as they became established and allowed the development of useful network applications, they quickly became indispensable. Data networks were (and still are) employed for the obvious uses such as resource sharing and reliability [BG92, Tan88]. The sharing of resources is important in two ways: cost and interface. If an organization requires an expensive computer or piece of hardware, it is much more cost effective to connect many users to that computer via inexpensive "dumb" terminals than to purchase each user the expensive computer. The same reasoning holds for expensive software that runs on the expensive computer systems. Again, it is much more cost effective to purchase additional software licenses than to purchase completely new copies of the software. In addition to the cost of the hardware and software, there is the human factor of interacting with said hardware and software — namely the interface. It is much easier for a human if, no matter which terminal is used in an organization to connect to the computer system, a constant set of interactions is presented. A common interface can also be considered as a reduction in cost, since training and maintenance costs can be reduced. As for reliability, it should be clear that having redundant hardware and software systems is imperative in certain organizations.

While these aspects of modern networking developed from the main-frame environment of the 1960's and 1970's, they apply equally well to a distributed data network environment [Lid90]. In addition to the economies of scale mentioned above in resource sharing, further economies became available in transmission bandwidth [BG92]. For example, while a 1.544 Mbps link contains twenty-four 64 Kbps chan-

nels, its cost is only a few multiples of that of a single 64 Kbps channel.[4] As well, transmission costs increase with distance, but at less than a one to one ratio. Hence data networks, proving themselves to be cost effective, created the need for applications which were network based. An organization with a geographically distributed operation requires networks in order to coordinate its operations. Thus, for example, the need for remotely updating a data base gave rise to the field of distributed systems [CD88]. Another example is the now common automated teller machine, or banking machine. The interconnection of computers also lead to the development of completely new applications, such as electronic mail, facsimile, and those that make use of audio and video [PT90, WT90].

1.1.2 The Emergence of ISDN

With data networks firmly entrenched, and network applications requiring more and more bandwidth, in 1984 CCITT adopted the recommendations for the Integrated Services Digital Network (ISDN) [BAF+88, Bla95, Onv94, SHP91, Tan88]. ISDN represents a world-wide attempt to replace the existing analog telephone networks with a digital system. In fact, ISDN is based on a digitized telephone network, that is 64 Kbps channels. Because of this, it is inherently a circuit switched network which also allows packet switching. Thus, ISDN is envisioned to support a broad range of voice and non-voice services, such as images and video, by integrating

[4]Note, however, that for the reasons of large distances and sparsely populated areas, this does not hold true in Canada. For example, in the United States, a trunk line from New York to Los Angeles would pass through many urban centers containing many users who may wish to use a portion of the trunk covering only a small distance. Thus the cost of the trunk line is borne by more users than just those located at the end points of the line. Compare this with a trunk line from Toronto to Vancouver, which passes through only a few (relatively small) urban centres.

telephone and data networks. In other words, ISDN combines the functionality of circuit switched and packet switched networks.

Two access interfaces are defined: basic and primary. The basic service consists of two 64 Kbps data channels and a 16 Kbps signaling channel, for a total of 144 Kbps. The primary service consists of twenty-three 64 Kbps data channels and a 64 Kbps signalling channel, for a total of 1.544 Mbps.[5] The data channels can be used together, in order to form higher bandwidth connections. Thus, ISDN supports multiple independent channels which are interleaved using Synchronous Transfer Mode (STM).[6] Unfortunately, even with a primary service bandwidth of 1.544 Mbps, video and image applications are poorly supported by ISDN. This observation is aggravated by the fact that the primary service is intended for business users, and the basic for home users. With only 144 Kbps available, broadcast quality video is intolerable. Hence, the emergence of Broadband-ISDN (B-ISDN).

1.1.3 ISDN leads to B-ISDN and ATM

B-ISDN is being developed to support high bandwidth applications such as image retrieval and video, as mentioned above, and also the interconnection of Local Area Networks (LANs) [BAF+88, BG92, Oht94, Onv94, SHP91]. Like ISDN, B-ISDN is intended to be an "all-purpose" network. As such, B-ISDN must be able to support: traffic streams with a large range of characteristics, such as constant or

[5] In Europe, primary service consists of thirty 64 Kbps data channels and a 64 Kbps signalling channel, for a total of 2.048 Mbps. This is due to the fact that the European telephone networks are based on an E1 scheme, as opposed to the T1 scheme in North America (and Japan). Henceforth, only the T1 scheme is considered.

[6] For a discussion of STM, refer to [BG92, Tan88].

variable bit rates and bursty traffic; a variety of network configurations, such as connection-oriented or connectionless links, point-to-point or point-to-multipoint connections; and a wide variety of transmission rates, with the ability to guarantee service levels. Since its links are optical fiber, B-ISDN has the required bandwidth and low bit error rates to fulfill the aforementioned intention.

It is amusing to note that there are about as many meanings for the term broadband in computer networking as there are people using the term. In B-ISDN, broadband refers to "a lot of bandwidth," which should be taken to mean more that 2 Mbps. Hence by employing optical fiber, B-ISDN is able to satisfy the needs of bandwidth-hungry applications.

With plans for B-ISDN systems to employ links with transmission rates between 100 Mbps and 600 Mbps, questions arose as to the wisdom of simply "beefing up" the existing STM-based ISDN system. STM employs circuit switching, and because of this does not benefit from statistical multiplexing. Since many of the current applications that will use B-ISDN networks, such as voice, have low bit rates when compared to the proposed link rates, much of the bandwidth allocated to a low rate connection would be wasted. In addition, the smallest rate available to an application using this STM-based network is 64 Kbps, which is very inflexible. Continuing the example of a voice connection and considering advances in voice coding [Hay88], for example by using Adaptive Sub-band Coding (ASBC), only 16 Kbps is required for this connection [Dau82, Jay86].

Therefore, in order to increase bandwidth utilization, and for reasons of switching and local loop complexity, STM was abandoned in favor of an *asynchronous*

packet switching mode. In this context, asynchronous refers to the fact that user information can appear at irregular intervals over a network link, or in other words information is transmitted over the network based on the needs of user applications, not on timing considerations of the network. And thus for brevity, not mentioning many considerations taken by the standardization bodies, ATM[7] was chosen as the transport, network and data link layer [BG92, Tan88] protocol for B-ISDN. Since ATM is to replace the functionality of STM in ISDN, its standard specifies a physical layer, namely the Synchronous Optical Network (SONET) [BC89].

At this point, this brief history of modern networking comes to a close. Further discussion on general technical aspects of ATM, such as signaling and framing with respect to SONET, while important, are not germane to this work. For more information, the reader should refer to [BG92, Bla95, Com94, Com95, HHS94, MS95, Par94] Nonetheless, a more detailed overview of certain aspects of ATM which are important when considering traffic classification and shaping is given in the following section.

[7]ATM has also been referred to as Fast Packet Switching (FPS) and Dynamic Time Division Multiplexing (DTDM).

1.2 Asynchronous Transfer Mode (ATM)

This section focuses on the aspects of ATM which are pertinent to the work of this thesis. Recall from the previous section that since ATM is the protocol which is used to implement B-ISDN, it inherits many of the properties of B-ISDN. ATM is a high speed, connection-oriented, packet switched data network, intended to operate over optical fiber links which have inherently low bit error rates. While it is connection-oriented, it supports connectionless services, and due to its asynchronous packet-oriented multiplexing behavior, it is well suited to bursty applications. ATM integrates the features of packet switched and circuit switched networks [BG92, Bla95, Com94, HHS94, MS95, Oht94, Onv94, Par94, Sai94].

Section 1.2.3 introduces the User-Network Interface, which specifies how connections are made to an ATM network. Then, the important features of the User-Network Interface, as they pertain to traffic classification and shaping, namely Call Admission Control and Usage Parameter Control are discussed in Sections 1.2.3.1 and 1.2.3.2, respectively, followed by an overview of traffic shaping in Section 1.2.3.3. Before discussing these aspects of ATM, it is important to be aware of two of the driving forces behind them: statistical multiplexing and its side effect, congestion, discussed in the following section, and Quality of Service and the Traffic Contract, discussed in Section 1.2.2.

1.2.1 Statistical Multiplexing and Congestion

As mentioned in Section 1.1.1, in order to more efficiently use the available bandwidth, statistical multiplexing is performed. Statistical multiplexing is said to

occur when two or more connections use a link in such a way that the link capacity is *less* than the aggregate peak capacity requirements of the connections [Jai90, MS95, Par94, Sai94]. Note then, as Figure 1.1 shows, that if all the connections desire to use the link at the same instant, not enough capacity is available. This occurrence is referred to as burst-level congestion,[8] or simply congestion. The

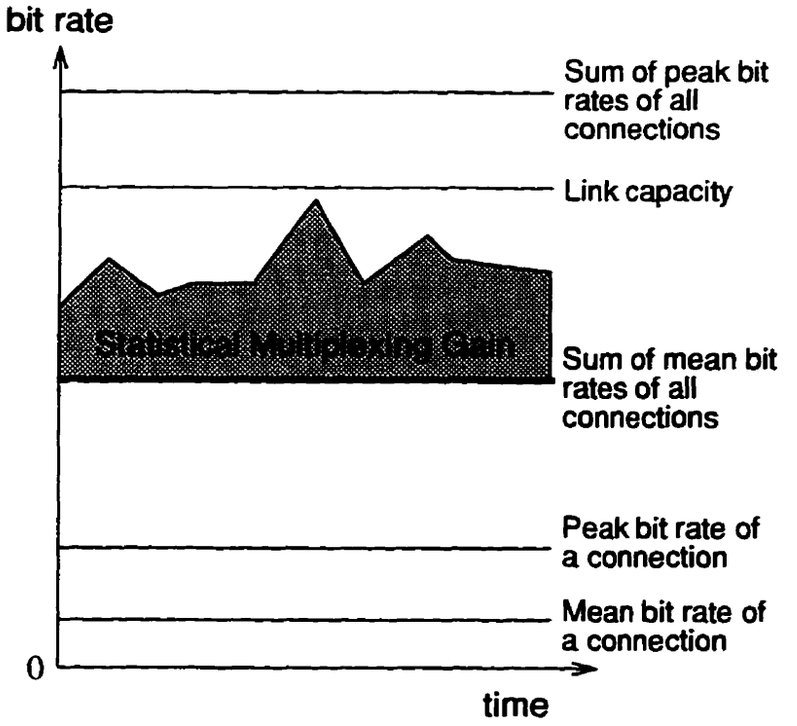


Figure 1.1: Statistical Multiplexing Gain

label *statistical* in statistical multiplexing, hence, indicates that this occurrence is unusual. That is, *on average* congestion will not occur, since the average aggre-

[8]Contrast this to call-level and cell-level congestion [Hui88]. While important, call-level congestion is not considered here.

gate rate of the network connections is less than the link capacity, as Figure 1.1 shows. Statistical multiplexing is essentially the difference between STM and ATM. In STM bandwidth is assigned to a given connection, even if there is no data to transmit, whereas in ATM there is no bandwidth assignation, and so a connection uses the link only when it requires.

Even with the large amount of bandwidth available in ATM networks, congestion must be controlled because as previously stated, ATM guarantees a certain level of service to connections. Note that congestion may also occur even if there is ample bandwidth available for the connections, as Figure 1.2 shows, due to *cell-bunching*, referred to as contention. In this case, two or more connections desire to

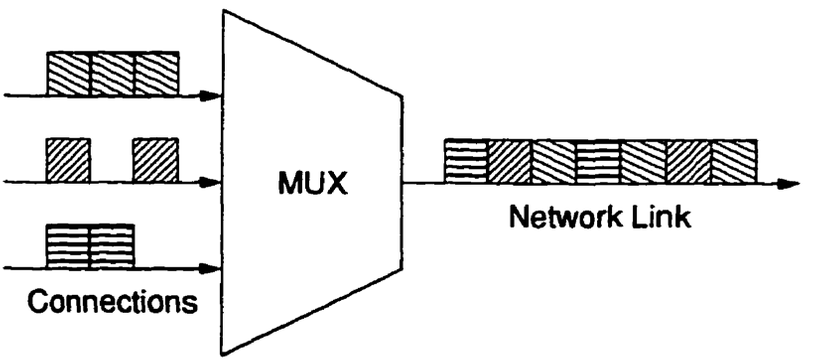


Figure 1.2: Cell-level Contention at a Multiplexer (MUX)

transmit at the same point in time, and even though there is sufficient link capacity for all the connections, only one can use the link at a given time. This, of course, is due to the fact the that a multiplexer is a multiple-access device [BG92, Tan88].

In the literature, there are in general two ways to solve the problems associated with congestion: preventive congestion control and reactive congestion control

[WRR88]. Preventive congestion control manifests itself in the User-Network Interface of ATM networks as admission control, and reactive as bandwidth enforcement. These methods will be discussed in Sections 1.2.3.1 and 1.2.3.2. Both of these, however, require knowledge of not only the traffic stream that is to be admitted to the network, but also of the existing connections. Thus, before a connection can take place, the user and network provider must agree upon the characteristics of the source and also the Quality of Service the user can expect from the network, which takes the form of a Traffic Contract, discussed next.

1.2.2 Quality of Service and Traffic Contracts

Section 1.1.3 stated that one of the important features of B-ISDN, and thus ATM, is the ability for the network to guarantee a level of service to the user. In the language of ATM networks, this is referred to as Quality of Service (QoS) [Com94, Kur93, LP91, MS95]. Some examples of service guarantees include cell loss rate, minimum bandwidth and maximum end-to-end delay. While much work has been performed and is ongoing concerning methods to guarantee a level of QoS, for this work it is sufficient to realize that source information is required to achieve this guarantee.

A number of QoS *classes* have been defined, such as best effort, constant bit rate, variable bit rate, connection-oriented and connectionless. QoS classes are of great importance in an ATM network, since it is intended to carry a wide variety of traffic streams as a result of service integration. A data source, for example, is extremely loss sensitive but delay insensitive. On the other hand, a voice source

is more sensitive to delays than losses. And video sources are both delay and loss sensitive. Thus, different actions and thus service guarantees are required when different applications wish to connect to the network. This information is exchanged via the *Traffic Contract.*

A Traffic Contract represents an agreement between the network provider and a user of the network. It specifies the QoS of the connection, the characteristics of the traffic stream, rules for determining whether a connection is complying with the Traffic Contract, and the definition of compliance itself. The importance of the Traffic Contract here is the requirement for characteristics of the traffic stream, and thus a method to obtain these characteristics. Since Traffic Contracts are negotiated at the User-Network Interface, it is described next.

1.2.3 Overview of the User-Network Interface

While much research has been performed and is continuing concerning the internal operations of an ATM network, such as switching, routing and management functions to name a few, the focus of this work occurs at the User-Network Interface (UNI).[9] Figure 1.3 gives a simple overview of this feature of an ATM network. It is generally assumed that due to the relatively high bandwidth of ATM links, compared to most existing applications, that multiplexing will occur at the source of Figure 1.3 (and demultiplexing at the destination). For example, a group of terminals, user applications, or LANs are multiplexed to share access to the ATM

[9]Note that ATM networks have been specified in many ATM Forum documents, see `http://www.atmforum.com`, but the document most applicable to this work is the User-Network Interface (UNI) Specification, Version 3.1 [Com94].

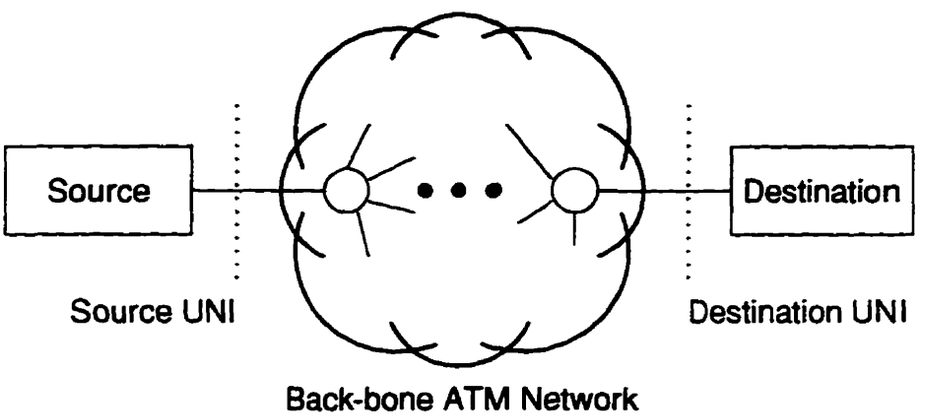


Figure 1.3: A General ATM Network, showing Source and Destination

network. Of course, an application with high bandwidth requirements may be connected directly, but for the present this is unlikely. Inside the back-bone ATM network, two network service providers may need to interface with each other. This point is called the Network-Network Interface (NNI), but does not concern this work.

Zooming in on the source UNI, then, Figure 1.4 depicts the portions of the UNI which are germane to this thesis. As the figure shows, the UNI consists of three basic elements: the Call Admission Control (CAC) function, the Usage Parameter Control (UPC) function, and the ATM switch, which performs cell-level, possibly priority-based scheduling. In addition, a fourth optional element is that of traffic shaping, with possible locations S1–S4 as shown in Figure 1.4. A discussion of traffic shaping is deferred to Section 1.2.3.3. The existing network connections are shown by the shaded ellipse, and the single call discussed in this overview is shown at the bottom of Figure 1.4.

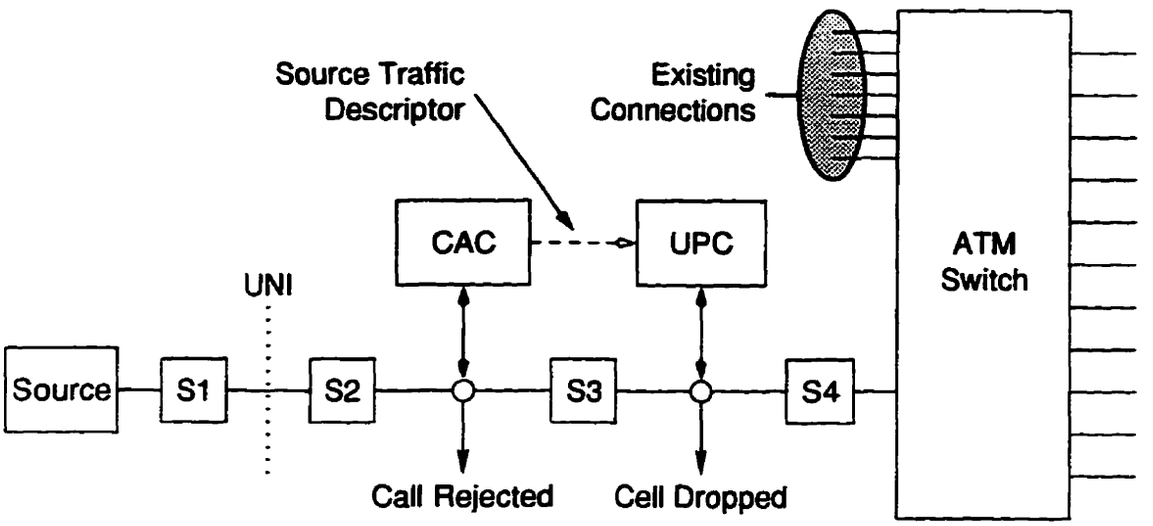


Figure 1.4: Overview of the ATM User-Network Interface (UNI)

There are two basic functions that occur at the UNI. First, the network provider must make a decision to either accept or reject the call, which is performed by CAC. Second, once a call is accepted to the network, this connection must be monitored to ensure compliance to the Traffic Contract, which is performed by UPC. It is important to state again that both CAC and UPC require accurate and timely source characteristics in order to perform their respective functions. While the concepts of QoS, CAC and UPC are usually treated as separate entities in the literature, they are nonetheless tightly intertwined by their mutual requirements for traffic stream characteristics. The requirements for these characteristics are emphasized in the following two sections.

1.2.3.1 Overview of Call Admission Control

CAC determines whether or not a connection may be admitted to the network. Since the network provider and the user "sign" a Traffic Contract at connection setup, the network guarantees the agreed upon QoS while the user makes a commitment to conform to a given traffic type, characterized by employing traffic parameters.

Traffic parameters include aspects of a traffic stream such as peak cell rate, sustainable cell rate, burst tolerance, but they may also be qualitative, such as a telephone or video source [Com94]. A subset of these parameters, the *source traffic descriptor*, can be used to describe the traffic stream characteristics of a given source. Once the connection request has been accepted, CAC has the source traffic descriptor, as well as routing and resource allocation information in hand.

In order to accept a connection, the network provider must make an estimation as to whether allowing the source access to the network would violate the source QoS and the QoS of existing connections. Given a source traffic descriptor, a number of methods have been suggested to accomplish this [DTVV90, ELL90, GAN91, GRF89, Hui88, JDSZ95, JV89, MSST91, Sai92, SS91, Tur92, WH91]. These methods include parametric models (statistical and fluid flow) and, or real time measurement based models, such as gaussian approximation, fast buffer reservation, class related rules, equivalent capacity and Sigma rule. While greatly enhancing network utilization by allowing statistical multiplexing, however, these methods either make assumptions about the source characteristics for tractability or require statistical measurements. As is discussed in Section 1.3, this creates difficulties

when performing CAC.

1.2.3.2 Overview of Usage Parameter Control

After a source has been admitted to the network, its data enters the UNI and the connection is monitored by UPC. The purpose of UPC is to ensure that the QoS of existing connections is not affected by the misbehavior of the source, this misbehavior being possibly malicious or unintentional [Com94]. Connection monitoring is performed by comparing the current traffic stream characteristics to the source traffic descriptor agreed upon during CAC and the negotiation of the Traffic Contract. Hence, as Figure 1.4 shows, the source traffic descriptor is passed from CAC to UPC. If the Traffic Contract is violated, cells are dropped.[10] Cells which conform to the specifications of the Traffic Contract are passed on to the ATM switch which, according to its scheduling scheme, duly injects the cells into the back-bone ATM network.

UPC usually takes the form of a *policer*, since it is the "enforcement arm" of the UNI. Some of the methods proposed for performing UPC include: the leaky bucket; jumping, sliding, moving and exponential windows; cell spacing; UPC flag cell; framing; and input rate regulation [BGSC92, BS91, GRF89, HH91, MGF91, OOM91, Rat91, TG92, Tur86, WRR88]. The drawbacks to most of these methods is discussed in Section 1.3. Just as the UNI has an inter-network analog, the NNI, mentioned in Section 1.2.3, UPC also has an inter-network analog, namely Network

[10]Note that, at the network provider's discretion, violating cells can be *tagged* and admitted to the network. If a tagged cell is present at a network node where congestion exists, it is dropped before any non-violating cells are dropped [Com94, Sai94].

Parameter Control (NPC). NPC performs policing on the traffic streams of network links. As with the NNI, NPC is not considered in this work.

1.2.3.3 Overview of Shaping

It is generally accepted that cell-space shaping, or simply shaping, is an integral part of the UNI in ATM networks [Com94]. Shaping is the act of interfering with individual cells of a traffic stream, such as delaying their transmission, to achieve a certain result.[11] Usually this is to reduce burstiness or Cell Delay Variation (CDV) and jitter [Com94, RVF91a], or to help define the source characteristics [Com94, MS95, Onv94, Par94, Sai94]. Shaping operates on a traffic stream with poorly defined or unknown characteristics with the purpose of defining them better. This has the triple effect of reducing the complexity of CAC and UPC schemes, since traffic characteristics are more readily available, and increasing the network efficiency at network multiplexing points, since cell-level congestion and contention can be mitigated.

Additionally, there is a consensus that traffic shaping should be applicable to a wide range of traffic types (ideally all traffic types), make traffic streams easier to describe at the UNI, and should simplify policing of a shaped stream. A final advantage of shaping is that given that a network user has knowledge of how the network provider characterizes and polices traffic streams, this allows the "pre-shaping" of streams so that Traffic Contracts are easier to negotiate and violations are minimized. There are two major drawbacks of shaping: first, buffering is required

[11] Of course, the cell order of a traffic stream must be preserved.

to perform shaping introduces delay into traffic streams; and second, additional complexity (hardware) is required for each stream connected at the UNI. However, these costs may seem small when compared to the reduction in complexity of CAC and UPC, and the benefits of increased network efficiency.

There are many proposed methods in the literature to perform traffic shaping [BGSC92, Bro92, BS91, Cha91, Gol90, GRF89, Kua94, MOSM89, Nie90, Nie93, OLT92, Rat91, RF91, RVF91b, SLCG89, Tur86, WM93, WRR88], which include: leaky buckets; token buckets; buffering; (r, T) smoothing; smoothing filters: cell-spacing; virtual scheduling; peak rate, source rate and burst length limiting; priority queueing; and framing. Perhaps unfortunately, much of the work on traffic shaping has also associated it with policing. This is easy to understand, considering that once an offending source is detected, something should be done to correct the situation. Thus many of the methods cited to accomplish policing in Section 1.2.3.2 are easily turned into traffic shapers.

It is the contention here that the roles of traffic shaping and policing should be separated. Therefore, a traffic shaper is a device which manipulates cell interarrival times to some end, but does not drop or tag any cells, whereas a policer is a device which monitors traffic streams, dropping or tagging cells as required, but which does not affect the cell interarrival times. This allows the shaping of traffic streams without the usual constraints associated with the cell loss rate. Further, if the shaper is placed at the output ports of switches in the interior of an ATM network, this could greatly simplify the scheduling function of downstream nodes. Mark that this is now feasible if the smoothing and policing functions are separated.

Current shaping devices, such as a leaky bucket, placed at the output of an ATM switch would additionally perform some sort of rate regulation, which may not be desirable.

There are a number of points along the route of a traffic stream through the network where shaping can take place, as shown by S1–S4 in Figure 1.4. The network provider may choose to perform shaping at any of these locations, perhaps multiple locations, or none. As stated in [Com94, section 3.6.3.2.5], "Traffic shaping is an optional function." Thus, there is some question as to the best location of the traffic shaper. While the position of the shaper is important, it is its operation which is the concern of a large portion of this thesis. Nonetheless, some pros and cons of locating the shaper at the four points specified are discussed next.

At position S1, the user's premises, the shaper is incorporated into the user equipment. It is unlikely that cells are dropped here, since the user most likely will purchase buffers large enough for applications, or perhaps employ a send-and-wait strategy. The network provider does not need to provide this buffering, and thus costs to the provider are reduced at the expense of the user. However, less buffering in the network has the effect of reducing network delays. In effect, the network buffers are "distributed" to the users' premises. If the user knows the exact nature of the cell buffering in the shaper, it is more likely that applications take this into account.

A very similar situation occurs at S2, which is at the UNI but before CAC. In this case, however, the network provider must supply the buffering. The bandwidth-delay aspect of high speed networks may be a problem if the distance from the

user to the UNI is large, and, or the source to UNI link is fast. A send-and-wait strategy cannot be used. More importantly, at both positions S1 and S2, the shaping is performed before CAC, and so the characteristics of a traffic stream will be translated into a possibly much different source traffic descriptor. This has a great effect on the operation of both CAC and UPC, as the previous sections implied. In effect, the shaper attempts to remove the fluctuations or burstiness from the traffic stream, and thus the operation of the UNI is simplified. However, the network is no longer transparent to the user, which is at odds with a basic premise of ATM networks.

Position S3, after CAC is performed, may at first glance be thought to be the same as S2, since after a source is connected to the network, CAC "disappears" from subsequent activities. However, position S3 represents a subtle difference in how traffic streams are perceived. In this case, traffic streams are observed during CAC unshaped, and so the methods of Section 1.2.3.1 operate on sources which most likely are difficult to characterize. Thus, if a source violates its Traffic Contract, the shaper may affect the traffic stream such that it returns to a state of conformance. In this case policing and thus UPC are very simple, and perhaps can be deleted. On the other hand, one may question the wisdom of shaping violating cells, since they should be dealt with by the policer.

Placing the shaper at position S4, after UPC (and CAC), appears to have limited appeal. Here, a network connection is already established and any violating cells have been dropped, and so the traffic stream is conforming to the traffic contract. Hence, no reductions in the complexity of CAC or UPC can be obtained. However,

since a shaper affects a stream at the cell level, it may be possible for the shaper to act as a *pre-multiplexer*. Since the ATM switch has knowledge of its scheduling scheme, contention problems that arise at the multiplexer of the switch can be reduced or even eliminated if the switch guided the shaping. That is, cells on each link to the switch can arrive in such a fashion as to alleviate contention and thus simplify scheduling. Similarly, the network provider may wish to shape network links in order to avoid congestion internal to the network. Thus an additional location, S5, could be placed on each outgoing links of the ATM switch of Figure 1.4.

Considering the pros and cons of each position, and the scope of this work, it is envisioned that shaping should take place at position S2. This allows reductions in the complexity of both CAC and UPC. In addition, it allows shaping and thus any improvements in shaping algorithms to be affected by the network provider. If a user wishes to by-pass shaping, this can be negotiated in the Traffic Contract. In any case, the effects of the shaper should be transparent to the user. That is, any delays introduced into the traffic stream by shaping should be removed at the destination UNI. This is one of the basic features of the Burst-oriented shaper. With the discussion of the major elements of the UNI complete, this brief overview of ATM comes to a close. The next section discusses some of the problems that occur at the UNI, which result due to the need for source characteristics.

1.3 Problems in CAC and UPC

It should be clear to the reader from the discussion of the previous section that the network provider requires information about the sources that wish to connect to the network so that Call Admission Control and Usage Parameter Control can be performed in order to guarantee a Quality of Service. The difference in ATM as opposed to traditional data networks is the broad range of services offered. This means that the User-Network Interface must be able to deal with a wide range of sources. For example, a Variable Bit Rate (VBR)[12] source may vary its transmission rate from a few kilobits per second to tens of megabits per second, and still be true to its source traffic descriptor. In contrast, another such as a Constant Bit Rate (CBR) source does not vary its transmission rate at all. Yet other sources, such as Available Bit Rate (ABR) and Unspecified Bit Rate (UBR) can vary their traffic type. Thus at the UNI, CAC and UPC must be able to handle sources with varying characteristics which are still in the same QoS class. It is these problems which lead to the work of this thesis, traffic classification and traffic shaping, about which more is presented in the following section and Section 1.3.2. Then, the goals of this work are stated in Section 1.4.

1.3.1 The Case for Traffic Classification

In general, the specification of source characteristics is not an easy task. The literature is replete with methods and models for characterizing sources and their parameters, some of which can be found in [CLG95, COO91, Des91, DM93, FMH93,

[12]Definitions of VBR, CBR, ABR and UBR sources can be found in [Com95].

FM94, Gus90, HL86, LTWW94, MAS+88, MH87, NFO89, Oht94, Onv94, RW90, SAG94, Sai94, Sch96, SMRA89, SW86, TGPM78, VP89, YS91], where much work has been performed in the measurement and modeling of data, voice and video sources. Unfortunately, it may not be possible to estimate source characteristics, especially if the source type is unknown (to the user). In this case it is the network provider's responsibility to characterize a traffic stream since a source traffic descriptor is required by CAC and UPC. In fact, since the network must perform UPC, it may be advisable for the provider to "second guess" the source characteristics, regardless of the source traffic descriptor given in the Traffic Contract. This would be especially useful in the case of misbehaving sources.

Note that if some the methods cited above are performed off-line, then the source information may not be accurate, and if performed on-line, then this information may not be timely. Since ATM networks operate at such high rates, short term source statistics may be important to these methods and thus congestion control. In general, however, statistical methods observe "longer term" trends in sources, thus potentially missing these "high frequency" source fluctuations. And of course, parametric or model-based methods do not allow for the possibility of new traffic types for applications or services not yet available.

Therefore there is a need for a method that can accurately and quickly determine the traffic class of a given traffic stream, without information of its statistics; this is a nonparametric approach. For the best range of application, it should be able to be used off-line when traffic streams are available before hand, and on-line when traffic streams are too non-stationary or unknown. In addition, for best performance of

the CAC and UPC, it should be able to identify short term fluctuations without losing sight of the longer term characteristics of the traffic stream. Herein lies the basic idea of this thesis.

The traffic classification method proposed can be thought of as a transformation operator, from the *statistical* domain to the *traffic primitive*[13] domain. That is, usually traffic streams are thought of in terms of their statistics, such as first and second order moments. Using the statistical tools of the above citations, various characteristics of sources can be observed, and thus the sources can be classified. Some of these classifications lead to familiar sources, such as geometric, Interrupted Bernoulli process (IBP) or Markov-modulated Bernoulli process (MMBP), to name a few. However, these "statistically well known" sources *are* well known simply because they provide tractability in analytical methods. Unfortunately, they do not appear with any regularity in most of the "real" [LTWW94] and anticipated ATM traffic streams.

The novel approach used in this work, however, does not rely on traditional statistical measures. Instead, traffic streams are considered to consist of a small group of well known traffic objects, termed traffic primitives. It is conjectured that these primitives contain information equivalent to the "traditional statistical information" of the methods of the citations. However, the information that traffic primitives provide is much easier to use, mainly because they can be observed on a relatively small time scale, that is cell-level as opposed to burst-level or call-level. Yet, by considering sequences of traffic primitives, longer term relationships can be

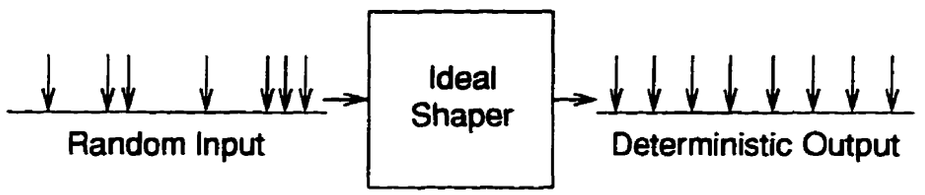
[13]Traffic primitives are the basic object of traffic classification, and will be described in detail in Chapter 2.

considered.

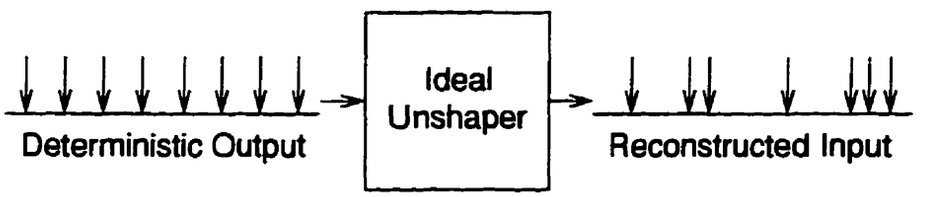
1.3.2 The Case for Traffic Shaping

Many of the reasons for performing traffic shaping at the UNI of ATM networks are mentioned in Sections 1.2.3.3 and 1.3.1. To reiterate, shaping is required so that unknown traffic streams can be characterized, and so that policing can be simplified. This shaping could take place at the the UNI, say at position S2 of Figure 1.4. It is interesting at this point to discuss the Ideal Shaper (IS) and Ideal Unshaper (IU) of Figure 1.5. The IS, shown in Figure 1.5(a), is a device which acts upon any traffic stream to produce a desired result, for example transforming a probabilistic stream into a deterministic one, but which is also transparent to the source and destination of the stream. Call the action that the IS performs on a traffic stream the Ideal Shaper function, denoted Γ. Now, if Ψ is the set of all traffic streams[14] and Ψ_D the set of all deterministic traffic streams that result after ideal shaping, and noting that $\Psi_D \subset \Psi$, then the IS performs the mapping $\Gamma : \Psi \longrightarrow \Psi_D$. This leads to a test of ideal shaping: if $\psi \in \Psi$, $\psi' \in \Psi_D$, and $\Gamma(\psi) = \psi'$, then the action of the shaper in question is ideal. As would be expected, the IU of Figure 1.5(b) performs the opposite action of the IS, in effect attempting to "undo" the effects of shaping the stream. If Γ^{-1} is the inverse of Γ, then it performs the mapping $\Gamma^{-1} : \Psi_D \longrightarrow \Psi$. Hence, ideally, $\Gamma^{-1}(\psi') = \psi$ is the action of the IU. Note that unshaping is not attainable using the usual shaping methods in the literature due to the fact that information such as intra-cell spacing is discarded.

[14]For a more complete definition of traffic streams, refer to Sections 1.6.5 and 1.6.6 contained in Section 1.6, Notation and Terminology.



(a) Creating a Deterministic Traffic Stream



(b) Reconstructing the Original Traffic Stream

Figure 1.5: The Ideal Shaper and the Ideal Unshaper

Of course, designing the ideal shaper is not statistically possible, but a *near* ideal shaper may be. If such a device is implemented at each user source to produce a near deterministic traffic stream, then the job of the multiplexer at the ATM switch would be trivial. It would perform *near deterministic multiplexing.* As discussed in the preceding sections, this is also beneficial for CAC, UPC and further downstream ATM switches. Similarly, designing the ideal unshaper is not possible either, since statistical information is removed from the unshaped traffic stream ψ in order to produce ψ_D. However, if this information is somehow stored within the shaped stream ψ_D, then an ideal unshaper could be approximated. This functionality is included in the proposed Burst-oriented shaper.

Some of the shaping algorithms mentioned in Section 1.2.3.2 have the problems

of reaction time and range of application. The reaction time of the leaky bucket and window algorithms is limited to the bucket size and window size, respectively. Other methods require statistical measurements of the traffic stream. Since these methods are usually designed to operate with any source type, they cannot benefit from knowledge of given characteristics of a specific source connected to the network. For example, if it is known that the leaky bucket is operating on a certain type of voice source, then the optimum (in some sense) leaky bucket buffer size may be, say, B_X. On the other hand, if the source is, say, a certain type of video stream, then it may be better if the leaky bucket buffer size was set to B_Y. The point is that by attempting to be applicable to all traffic sources, current methods cannot use specific knowledge of sources. However, if traffic streams could be identified, then the shaper could take action tailored to that stream. This feature is implemented in the proposed Variance Minimized shaper. Both of the shapers mentioned in this section, as well as traffic classification result from the goals of this research, which are stated in the following section.

1.4 Research Goals

The requirement for traffic characteristics in ATM networks has been emphasized in Section 1.3. It is the goal of this research to provide timely and accurate traffic information to the elements of the UNI by performing traffic classification and shaping. The following section discusses the motivation for using neural networks. Section 1.4.2 presents the goal of traffic classification, and Sections 1.4.3 and 1.4.4 present the goals of their respective traffic shaper. As opposed to the great amount of work in the literature that employs shapers to reduce CDV and jitter, the goal of both shapers here is to produce as deterministic a traffic stream as possible, that is to perform as close to the Ideal Shaper of Section 1.3.2 as possible. As a result, the functions of the UNI should be simplified.

1.4.1 Motivation

This work was begun with the intention of training a neural network to perform the actions of CAC, UPC, and shaping that take place at the UNI. Three neural networks would be trained, as shown in Figure 1.6 and loosely based on [Hir90], with the possibility of merging them into a single neural network, using established neural network techniques. However, as the work evolved, it was discovered that if attention was concentrated on shaping, then CAC and UPC would be simplified as a result of these efforts. This is true due to the fact that if it is possible to perform ideal shaping, policing would not be required since the source would always conform to its Traffic Contract (negotiated with shaping in mind).

Nonetheless, it should be noted that there has been quite an amount of work

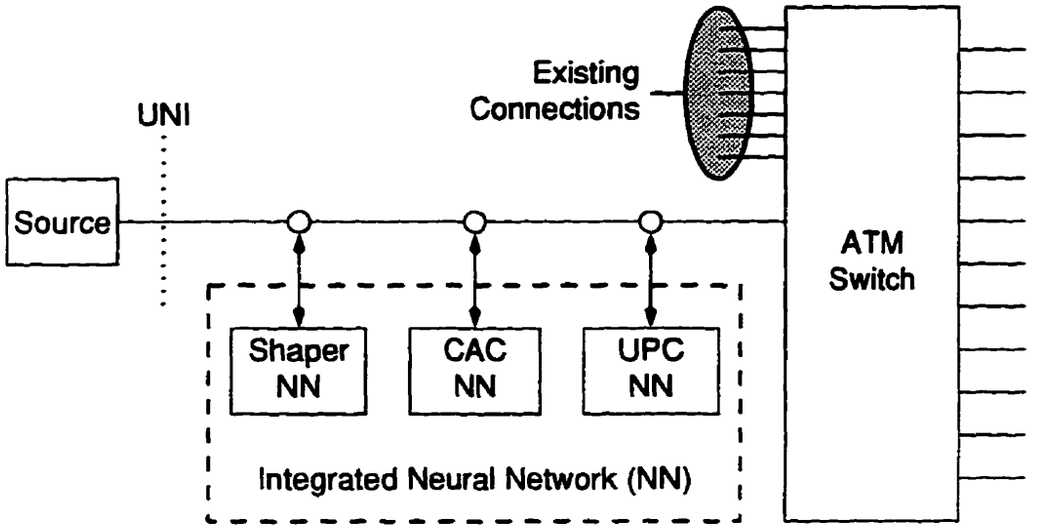


Figure 1.6: A Neural Network UNI Control Scheme

performed, some of which includes [CL91, Hir90, Hir91, LD97, TGG92, THS94], that attempts to tackle each component of Figure 1.6. Combining some of the functionality of these methods would not be a trivial task; but it would be possible. While the idea of having only one control mechanism at the UNI is appealing, drawbacks such as scalability and maintainability exist. The control scheme of Figure 1.6 should be considered further, but it is not a topic of this thesis.

Additionally, as work on traffic shaping progressed, it became apparent that it would be useful to be able to directly detect traffic types, instead of indirectly through a shaper. This would allow the CAC and UPC to operate without any dependence on traffic shaping. This is important since it is realized that ideal traffic shaping is not realizable, and also many sources, such as broadcast quality video, are intolerant to delays. Hence, the primitive classifier is developed, as well

as the Minimized Variance and Burst-oriented shapers.

1.4.2 The Primitive Classifier

The key idea of the primitive classifier is to treat a traffic stream as an object, or more precisely as a string of small, simple objects, instead of viewing it in terms of its statistics. With this in mind, one can devise a method that attempts to recognize these objects or patterns in the stream, which can then be used to classify and thus identify the stream. This is the underlying principle of the primitive classifier and thus the method of traffic classification proposed in this thesis. Aspects of work were presented at [LM96].

An ideal way to recognize patterns is to use neural networks. Neural networks have the ability to generalize, are noise tolerant, and are able to handle non-stationary data [RHW86, Hay94]. As well, once trained a neural network operates very fast, since it is a massively parallel device, and so its classifications should arrive in a timely manner even at ATM speeds. That is, compared to conventional (statistical) control mechanisms, all of the computational time that is required in estimating parameters is performed "up-front," during the off-line training of the neural network. In addition, if training is successful, these classifications are very accurate. Note that the current work is concerned with using neural networks to perform traffic classification, not with the theory of neural networks themselves.

1.4.3 The Minimized Variance Shaper

Once a traffic stream has been classified into a given traffic type using the primitive classifier, the Minimized Variance shaper (MVS) uses information specific to that source to attempt to shape the traffic stream to as nearly deterministic as possible. This is achieved by attempting to minimize the interdeparture time variance of the cells of the traffic stream exiting the MVS. That is to say, the inter-cell variance of a deterministic traffic stream is zero, and so this should be the goal of the shaper. Aspects of this work appear in [LM94].

1.4.4 The Burst-oriented Shaper

The Burst-oriented shaper (BOS) is developed from insights gained in the work on traffic classification. It attempts to isolate a traffic pattern that should be common in ATM networks, namely a period of contiguous cells followed by an idle period. It then simply attempts to spread the cells over the entire burst and idle period in a uniform fashion, again to achieve a deterministic output from the shaper. Since it contains some of the basic ingredients of traffic classification, it does not require knowledge of the traffic type, and should be considered an alternative to performing primitive classification.

An important feature of the BOS that sets it apart from existing shapers is that since it acts on a burst and idle period, it has knowledge of their respective lengths. By sending this information to the destination UNI which contains an *unshaper*, it is possible for the traffic stream to be presented to the destination exactly as it entered the network — barring any network problems. This has the desirable

effect of using shaping to alleviate the problems at the UNI while presenting the user with a network connection that is transparent. The next section gives a brief overview of neural networks as they pertain to this work. It should be reiterated that no contributions are made in the theory of neural networks; they are simply applied as a useful tool in order to perform traffic classification.

1.5 Overview of Neural Networks

Neural networks have been the topic of research for almost fifty years. They have been applied to problems such as adaptive control, pattern recognition and system identification, to name a few [Hay94, Lip87, MP88, Was89, WL90].[15] Enumerating these applications, however, would fill many pages and is not constructive for this work. Suffice it to say, neural networks have regained their initial popularity in the past decade.

The main advantages of neural networks is their parallelism and ability to generalize. Since they have many computational units which act in parallel, they operate very fast. Their ability to generalize means that they behave well in the presence of noisy inputs. If they are confronted with a situation that was not planned for in training, they will attempt to take the action that results in a similar situation for which they were trained. Thus, the idea that neural networks can learn.

Neural networks come in many flavors, such as competitive learning and self-organization, Adaptive Resonance Theory (ART), feature maps, Hopfield models, Bidirectional Associative Memory (BAM), and the Boltzmann Machine [Gro76a, Gro76b, Hop82, HS86, Koh82, Kos87]. However, arguably the most popular neural network paradigm is Backpropagation [RHW86], and it is employed in this work. Appendix A gives a full description of the operation and training of multilayered neural networks, and the Backpropagation training algorithm. The reader unfamiliar with neural networks may wish to refer to this appendix now.

---

[15] These few citations contain many references, and the reader is encouraged to consult them for more applications of neural networks.

The Backpropagation training algorithm is easy to implement, and under most circumstances convergence is probable. Unfortunately, as with most neural network paradigms used, there is no mathematical guarantee that convergence will occur in a finite amount of time. Another problem of the algorithm is the time required to train the network. The Backpropagation training algorithm is a supervised training method, which requires training vectors. A training vector is simply a way of describing to the neural network the problem which it must learn. It consists of an input to the neural network and its associated desired output. When an input is presented, the neural network output is compared to the desired output, the difference of which is used to update the weights of the neural network via the Backpropagation algorithm.

Neural networks have a wide range of application in communications in general [Hay94], as well as specifically in ATM networks , most notably in CAC, congestion control, routing and network design and management [CL91, DTW94, FDD97, Hir90, Hir91, KM91, LD97, Mor91, MTG93, OAT94, TGG92, THS94, YHS96]. These citations represent only a tiny sampling of the available publications. Thus the idea of using neural networks to learn ATM related input-output relationships is generally accepted in the literature.

1.6 Notation and Terminology

Terminology and notation, especially the free use of acronyms, can cause serious problems for the reader of technical subjects. This section attempts to prevent this situation by discussing or defining important aspects of what has been presented in previous sections, and also of what is to come. The reader may wish to examine the following sections now, or refer back to them as the need arises. Highlights include the definition of ATM cells and slots in Section 1.6.3, cellization in Section 1.6.4, traffic streams and the definition of sources in Section 1.6.6. Some of the sections are included for completeness, for example Section 1.6.5 on cell spacing. While these concepts and notation are not used directly in the chapters which follow, they were instrumental in developing the contributions of this thesis.

1.6.1 Notation

$\mathbb{Z}^+$: the positive integers; $\mathbb{Z}^+ = \{0, 1, 2, \ldots\}$.

$\mathbb{R}^+$: the positive reals; $\mathbb{R}^+ = [0, \infty)$.

$\mathbf{x}$: an emboldened variable represents a column vector.

1.6.2 Link Rate and Effective Rate

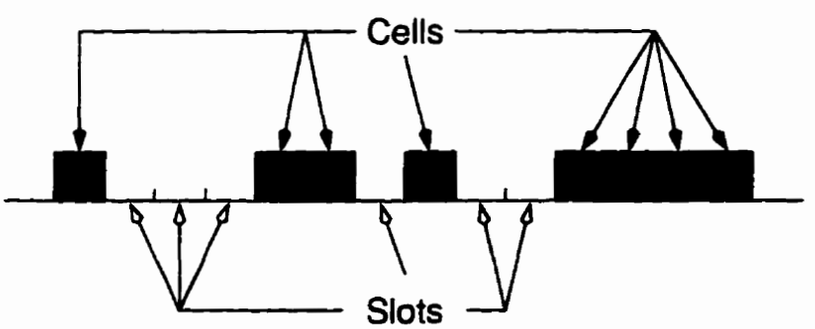
The link rate, C_L, is the transmission speed of the ATM backbone link, in bits per second (bps). It is the maximum speed at which data can be transmitted by the network provider. The effective rate, C, is the maximum speed at which a user may transmit data. Note that $C < C_L$ due to protocol overhead. Unless stated to

the contrary, the terms link capacity or simply capacity refer to the effective link rate.

1.6.3 ATM Cells and Slots

The ATM cell format is specified in [Com94, Section 3]. Here, however, it is sufficient to realize that an ATM cell, or simply cell, has a *fixed* size of 53 bytes; 48 bytes of data, and 5 bytes of header. Figure 1.7(a) depicts an ATM cell, where the header and data portion of the cell are drawn to scale. The exact structure of

(a) Overview of an ATM Cell



(b) Cells and Slots

Figure 1.7: The ATM Cell Layout and ATM Slots

the cell is not important to this work, rather the fact that the cell has a fixed size. The concept of a cell is so important to B-ISDN, and thus ATM, that it is defined

in ITU-T Recommendation I.113 [ITU91a] as "A cell is a block of fixed length. It is identified by a label at the ATM layer of the B-ISDN PRM."[16] In addition, ITU-T Recommendation I.321 [ITU91b] goes on to define cell types. Of these, at the ATM layer [Com94] are *assigned* and *unassigned* cells. However, delving into the standards to this degree does not help to clarify that which will be discussed in the chapters that follow. Thus, a generalized definition of a cell is given here that incorporates several aspects of ATM cells as they are defined in the standards, with the realization that this has a simplifying effect for the purposes of this thesis.

DEFINITION 1.6.1 (ATM CELL)

A cell is a unit consisting of 48 bytes of data. The time required to transmit a cell is

$$T_L = \frac{53 \times 8}{C_L} \quad seconds. \tag{1.1}$$

This transmission time corresponds to a cell transmission rate of $R_L = \frac{1}{T_L}$ cells per second. Note that due to the five byte overhead in an ATM cell, the highest possible throughput a user can achieve is $C = \frac{48}{53} R_L$ cells per second. Since an ATM cell has a fixed size, it is convenient to think of channel time in ATM links as slotted, with slot time T_L seconds. A slot can be assigned, in which case it contains a cell, or unassigned, in which case it is empty, as Figure 1.7(b) indicates. As mentioned, since the exact structure of the ATM cell is unimportant in this work, the ATM cells in Figure 1.7(b) no longer make the distinction between header and data; this convention will be used from this point.

[16] Protocol Reference Model.

The cell stream shown in Figure 1.7(b) consists of a cell followed by three consecutive empty slots,[17] two consecutive cells followed by a slot, a cell followed by two consecutive slots, and finally four consecutive cells. As can be seen, specifying streams of cells in this manner is very verbose, especially for long streams. Section 1.6.4 introduces a short form for specifying cell streams, and Section 1.6.6 gives a formal definition of streams of cells.

1.6.4 Cellization

Due to the small size of an ATM cell, it is unlikely that any applications will produce a traffic stream that is made up of cells. More likely, packets from applications or LANs will make up traffic streams. And since ATM is a packet switching network, a mechanism is required to break these larger data units into cells before they enter the ATM network, and then reconstruct them again as they leave. This functionality is performed by the ATM Adaptation Layer (AAL) [Com94], shown in Figure 1.8. Its pupose, in the context of this work, is that the AAL performs the function of *cellization*, coined here for ATM networks.

DEFINITION 1.6.2 (CELLIZATION)

Cellization is the process by which packets of data from users, applications and other networks is encapsulated into ATM cells.

In general, the packets will consist of header and data information, of variable length. These packets are expected to be much larger than an ATM cell. Cellization is analogous to packetization in traditional networks [BG92, Tan88]. It is convenient

[17] For the remainder of this thesis, the term slot is used to signify an empty slot.

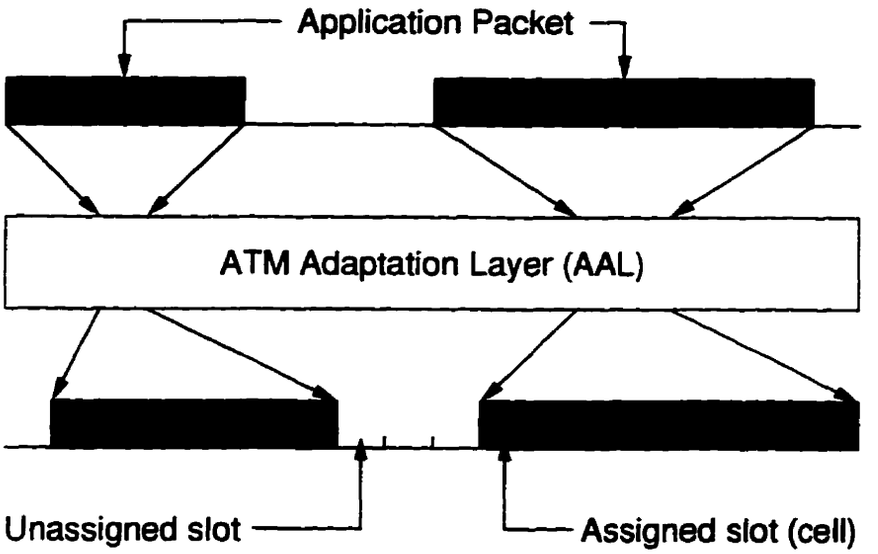


Figure 1.8: Traffic Partitioning at the ATM Adaptation Layer

then, after cellization, to represent a traffic stream as a string of "1's" or "0's," where a "1" indicates a cell, and a "0" a slot. With this notation, the cellized data of Figure 1.8 can be represented as "11111100011111111."

The salient feature of cellization, as it pertains to traffic primitive classification which is the subject of Section 2.1, is that a data packet is, most likely, transformed into a number of contiguous cells. Thus, a stream of packets is transformed into a group of cells followed a group of slots, and so on as the traffic stream passes through the AAL. In other words, the traffic stream departing the AAL can be described as a packet train [JR86]. This fact is paramount in the definition of the traffic primitives.

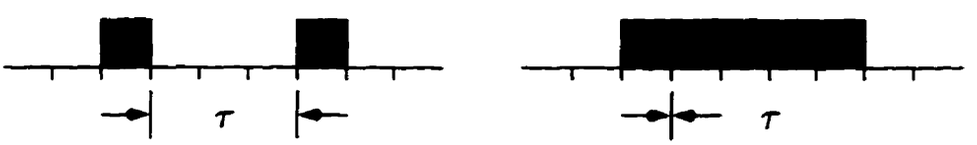
1.6.5 Cell Spacing and Cell Bursts

In order to aid in the development of traffic classification and shaping, it is necessary to define cell spacing. This section and portions of the following are included for completeness, so that the reader has some indication of the path followed in this work.

DEFINITION 1.6.3 (CELL SPACING, τ)

Cell spacing, or intra-cell spacing, τ, is the number of consecutive slots between two cells.

Note that since there can be zero or more slots between two cells, $\tau \in \mathbb{Z}^+$. Figure 1.9 gives two examples of cell spacing. Figure 1.9(a) shows a case where the intra-



(a) $\tau = 3$ (b) $\tau = 0$

Figure 1.9: Two Examples of Cell Spacing

cell spacing is $\tau = 3$, whereas Figure 1.9(b) shows the case for a burst, that is $\tau = 0$. This example leads naturally to the next definition.

DEFINITION 1.6.4 (BURST)

A burst is the situation that occurs when the cell spacing between two cells is zero. that is $\tau = 0$.

A result of this definition is that it requires two or more consecutive cells to create a burst; a single cell is not a burst.

1.6.6 Traffic Streams

The definition of cell spacing of the previous section is used here to define a traffic stream, and this definition can then be used to specify different types of traffic streams, such as constant bit rate, packet train and on-off, as is done in Sections 1.6.6.1–1.6.6.3. Finally, a more general traffic stream definition is given in Section 1.6.6.4, which forms the basis of the description of traffic sources used in the following chapters.

As a traffic stream is passed to the AAL, it experiences cellization into ATM cells, as discussed in Section 1.6.4. As such, it can be described as a finite sequence of cells separated by a possibly time varying but undoubtably random cell spacing, τ_i, as introduced in Section 1.6.5. Here, i denotes the length of the i^{th} intra-cell time, in slots, between the i^{th} and $(i+1)^{\text{st}}$ cells in the stream. Therefore, it is possible to describe this traffic stream not in terms of its cell arrival times, but rather by a finite sequence of its intra-cell spacing.

For example, consider Figure 1.10, which is the same cell stream as in Figure 1.7(b). It depicts a traffic stream of eight cells and six slots. Using the above

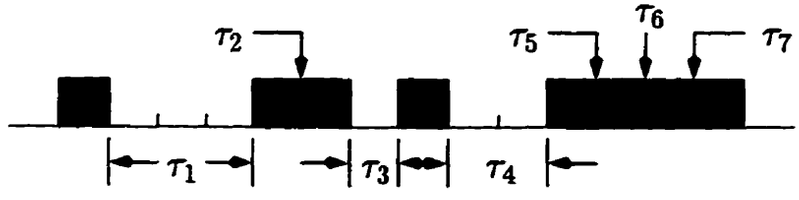


Figure 1.10: Specifying a Traffic Stream

notation, this stream can be described by the cell spacing sequence $\tau_1 = 3$, $\tau_2 =$

$0, \tau_3 = 1, \tau_4 = 2, \tau_5 = 0, \tau_6 = 0, \tau_7 = 0$, or equivalently $\psi = \langle 3, 0, 1, 2, 0, 0, 0 \rangle$, where ψ is a cell spacing sequence variable. More generally, let

$$\langle \tau_1, \tau_2, \tau_3, \ldots, \tau_N \rangle \tag{1.2}$$

be a finite sequence of N elements, $\tau_i \in \mathbb{Z}^+$, $N \in \mathbb{Z}^+$, and let Ψ denote the set of all such finite sequences, that is

$$\Psi = \left\{ \text{all finite sequences whose elements } \tau_i \text{ are drawn from } \mathbb{Z}^+ \right\}. \tag{1.3}$$

This leads to the following definition.

DEFINITION 1.6.5 (TRAFFIC STREAM)
A traffic stream ψ is a particular realization of a finite cell spacing sequence drawn from Ψ, that is $\psi \in \Psi$.

With this definition, the notation ψ^N denotes a traffic stream with N cell spacing elements. Using this notation, it is possible to define some common traffic streams, as follows.

1.6.6.1 Deterministic or CBR Streams, Ψ_D

A deterministic or Constant Bit Rate (CBR) stream is one which does not have any variation in the intra-cell spacing; that is, it has no random component. Refer to Figure 1.11, which gives an example of a CBR stream utilizing 50% of the link capacity, C. Since the term CBR is used most commonly in the literature to denote

Figure 1.11: An Example of a Constant Bit Rate (CBR) Traffic Stream

a deterministic source, the term deterministic stream will no longer be employed.

DEFINITION 1.6.6 (CBR STREAMS, Ψ_D)

CBR traffic streams, $\Psi_D \subset \Psi$, is the set of all finite sequences such that each element of the cell spacing sequence has the same value, that is

$$\Psi_D = \{\textit{all finite sequences with elements } \tau_i{=}K \in \mathbb{Z}^+\}\,. \tag{1.4}$$

The particular realization of the CBR traffic stream of Figure 1.11 can be described as 101010101010101, or $\langle 1,1,1,1,1,1,1\rangle$, using the present notation. More generally, any sub-multiple of the link capacity can be given by

$$\psi_D^N(K) = \langle K, K, \ldots, \tau_N = K\rangle\,, \tag{1.5}$$

which represents a CBR traffic stream, length $N(K+1)+1$ cells, with a cell arrival rate $\frac{C}{K+1}$ cells per second.

1.6.6.2 Packet Train (PT) Streams, Ψ_{PT}

A packet train (PT) stream, like the CBR stream, does not contain any random component. Cells arrive in well defined groups, or packets, with equal spacing within and between the groups. The number of cells in each group is the same, as

is the number of slots between each group. The example depicted in Figure 1.12 shows a PT stream, where two cells arrive side by side, followed by two slots, and can be written as $\langle 0,2,0,2,0,2,0\rangle = 11001100110011$. It also utilizes 50% of the

Figure 1.12: An Example of a Packet Train (PT) Traffic Stream

link capacity. This leads to the following definition.

DEFINITION 1.6.7 (PT STREAMS, Ψ_{PT})

PT traffic streams, $\Psi_{PT} \subset \Psi$, is the set of all finite sequences such that at least two contiguous cells are followed by at least one slot, repeating, that is

$\Psi_{PT} = \{$*all finite sequences with elements*

$$\tau_1, \tau_2, \cdots, \tau_i, \tau_j, \tau_{j+1}, \tau_{j+2}, \cdots, \tau_{i+j}, \tau_{2j}, \tau_{2j+1}, \tau_{2j+2}, \cdots, \tau_{i+2j}, \tau_{3j},$$

$$\tau_{3j+1}, \tau_{3j+2}, \cdots, \tau_{i+3j}, \cdots$$

such that

$$\tau_1 = \tau_2 = \cdots = \tau_i = \tau_{j+1} = \tau_{j+2} = \cdots = \tau_{i+j} = \tau_{2j+1} = \tau_{2j+2} = \cdots = \tau_{i+2j}$$

$$= \tau_{3j+1} = \tau_{3j+2} = \cdots = \tau_{i+3j} = \cdots = 0,$$

and $i \geq 1$, $j = i+1$, $\tau_j = K > 0$, $K \in \mathbb{Z}^+\}$. (1.6)

Note that for a stream to be considered a PT stream, it must have at least two cells side by side, followed by at least one slot, and that it ends with a burst of cells.

From this definition, a general PT stream can be written as

$$\psi_{PT}^{N}(i,K) = \langle 0,0,\ldots,\tau_i = 0,\tau_{i+1} = K,\ldots,\tau_N = 0\rangle\,, \tag{1.7}$$

which represents a PT traffic stream with burst size $i+1$ cells in length and idle size K slots in length, for a total stream length of $\left(\frac{N-i}{i+1}\right)(K+i+1)+i+1$ cells, and a cell arrival rate of $\frac{(N+1)(i+1)C}{(N+1)(i+1)+K(N-i)}$ cells per second.

1.6.6.3 On-Off Streams, Ψ_{OO}

An on-off stream has the same overall structure as the packet train, in that groups of cells arrive together, however the number of cells in a group, as well as the intra-group and inter-group spacings are random. The randomness, of course, is defined by some underlying probability distribution. Since it is difficult to show a distribution with only a few cells, Figure 1.13 gives an example of an on-off stream whose "on-period" is deterministic of size six cells, whose "off-period" is

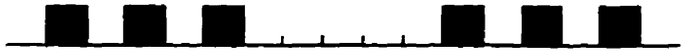


Figure 1.13: An Example of an On-off Traffic Stream

deterministic of size four slots, and with the on-period distributed as the CBR stream of Figure 1.11. This stream utilizes 40% of the link capacity.

DEFINITION 1.6.8 (ON-OFF STREAMS, Ψ_{OO})

The set of On-off traffic streams, $\Psi_{OO} \subset \Psi$, is a set of all finite sequences such that an on-period is followed by an off-period, repeating. Define Ω to be the

set of all possible discrete probability distributions, including the deterministic "distribution." The length of the on-period is a random variable taken from some distribution in Ω, and the length of the off-period is a random variable taken from a possibly different distribution in Ω. Within the on-period, the inter-cell spacing is a random variable taken from a possibly third distribution in Ω. Once the three distributions have been chosen, they are invariant.

As can be seen from the previous two sections, this cell-level notation is not amenable for traffic streams with elements taken from probability distributions. The stream notation is used for developing the primitive classifier and traffic shapers, and is not intended for this level of complexity. Thus, as is introduced in the next section, a more efficient notation can be used, based on this definition of an on-off traffic stream.

1.6.6.4 General On-off Traffic Sources

Pursuant to the discussion of the previous section, the definition of a general on-off traffic source follows. It can be used to characterize all of the sources of traffic streams mentioned in the previous sections, namely CBR, PT and, of course, on-off.

DEFINITION 1.6.9 (ON-OFF TRAFFIC SOURCE, $\Psi_{OO}(\varpi, \varsigma, \varphi)$)
An on-off traffic source, $\Psi_{OO}(\varpi, \varsigma, \varphi)$, is a traffic source of alternating periods where the source is either "on" or "off," as shown in Figure 1.13. The distribution of the length of the on-period is given by ϖ and the distribution of the off-period by ς;

the inter-cell distribution within an on-period is given by φ, where

$$\begin{aligned} \varpi, \varsigma, \varphi \in & \{\textit{all possible discrete distributions, including deterministic}\} \\ & = \{\mathcal{D}(x), \mathcal{B}(p), \mathcal{G}(p), \mathcal{U}(x), \mathcal{H}(\mathbf{x}, \mathbf{p}), \ldots, \emptyset\} \\ & = \Omega, \end{aligned} \tag{1.8}$$

where

$\mathcal{D}(x)$: indicates a deterministic distribution, where the outcome is always x.

$\mathcal{U}(x)$: indicates a uniform distribution, where the outcome is uniformly distributed between 1 and x,

$\mathcal{B}(p)$: indicates a Bernoulli distribution, with parameter p,

$\mathcal{G}(p)$: indicates a Geometric distribution, with parameter p,

$\mathcal{H}(\mathbf{x}, \mathbf{p})$: indicates a general discrete distribution, where the probability of outcome $x_i \in \mathbf{x}$ is given by the corresponding $p_i \in \mathbf{p}$,

$\emptyset$: indicates the null distribution, which has no outcome.

Note that if a certain distribution is not required, for example a CBR source does not have an off-period, then the convention is to use the null distribution, which implies that ς is not required.

Using this new notation, the CBR source of section 1.6.6.1 can be represented by $CBR(x) = \Psi_{OO}(\mathcal{D}(1), \mathcal{D}(x), \emptyset)$, which utilizes $\frac{1}{x+1}$ of the link rate. Thus, the 1010101 source is given by CBR(1) and utilizes $\frac{1}{2}$ of the link rate. Similarly, a

1001001 source can be represented by CBR(2). The packet train source of Section 1.6.6.2 is, in general, given by $\mathrm{PT}(x,y) = \Psi_{OO}(\mathcal{D}(x), \emptyset, \mathcal{D}(y))$. Thus, the PT source of Figure 1.12 can be expressed as $\mathrm{PT}(2,2)$. Since on-off sources are considerably more complicated, they will be defined as required.

With a generalized on-off source defined, this section on notation draws to a close, as does the Introduction. The following chapter describes a major contribution of this thesis, the primitive classifier.

Chapter 2

Traffic Classification

The novel idea of the traffic classification method proposed is to view traffic streams as collections of objects, which represents the core of this research. As introduced extensively in Chapter 1, traffic classification is required in order to determine source characteristics.

In this chapter, Section 2.1 introduces the concept of the traffic primitives, and discusses both the characterization of general traffic streams and their partitioning into broad groups. Doing so allows the specification of traffic primitives and finally the traffic primitive classifier. This section closes with a brief word on the scalability of this neural network based traffic classification method by briefly mentioning the compound classifier. In Section 2.2, the training methods employed are discussed, and as well the validity of the training. Section 2.3 shows the results of training three different primitive classifiers using 10–35–35–9, 15–80–80–10 and 20–200–200–11 neural networks. The remainder of this chapter, Section 2.4, discusses the results of operating the primitive classifier with traffic streams not presented to the neural

networks during training. This shows a neural network's ability to generalize, and thus the wide range of applicability of this method.

2.1 Classification of Traffic Primitives

Traffic classification is based on the premise that a traffic stream can be broken down into a few basic traffic objects, called *traffic primitives*. When these primitives are arranged in certain patterns, they can describe more complicated sources. Thus, with the correct choice of the ordering of strings of traffic primitives, it should be possible to create a "recipe" which describes how to reproduce a given stream. If one considers a wide range of traffic streams, they invariably consist of groups of data and idle periods interspersed in some manner. Thus, the traffic primitives should be chosen to simply mimic this observation. In an ATM setting, groups of data are represented by cells, and idle periods by slots, as defined in Section 1.6.3.

The following Section 2.1.1 discusses how streams of data, or traffic streams, can be observed and characterized so that features of the stream can be detected. Once certain features of an unknown traffic stream are observed, they can be compared to those of known traffic streams. In this way, unknown sources can be classified. While this technique is not new, the novel approach of this work employs neural networks to perform the feature comparison and thus classification. Section 2.1.2 takes the insights gained by considering the characteristics of traffic streams and applies them specifically to the cell streams which exit the ATM Adaptation Layer. Due to the defined behavior of cells and slots after cellization,[1] this knowledge can be used to partition traffic streams into basic groups, or traffic types. From these, as mentioned above, it is envisioned that more complicated traffic streams can be classified. The characterizing and partitioning of traffic streams will form the basis

[1]Cellization is defined in Section 1.6.4.

for the definition of traffic primitives in Section 2.1.3. Section 2.1.4 then introduces the neural network based classifier which will detect features of an unknown traffic stream by comparing them to features of streams it has been trained to recognize. The classifier then outputs which traffic primitives it recognizes in the stream. Section 2.1.5 makes an analogy between traffic primitives and Optical Character Recognition, and finally Section 2.1.6 discusses the scalability of the classifier by introducing the Compound Primitive Classifier.

2.1.1 Characterizing Traffic Streams

In this section, an atypical traffic stream will be examined in order to discuss various characteristics of ATM traffic. This somewhat contrived stream, which could be a video source, for example, is shown in Figure 2.1. It is simply devised to help

Figure 2.1: Identifying Primitives in a Traffic Stream

explain the concept of traffic primitives, since it is rich in the types of features upon which the primitive classification is based. The stream of Figure 2.1 can be subdivided into three sections, labeled A, B and C, where in each section the stream takes on a certain "character." One could say that in section A the stream more or less is "all on," that is, it utilizes the entire transmission link capacity. In section B, its character is that of a constant bit rate source that utilizes half of the link

capacity. Lastly, in section C the stream resembles a packet train source, utilizing about 40% of the link. Depending on the amount of the stream an observer wishes to consider, the boundaries of these three sections — and thus the character of the traffic stream — may change. For example, by shifting the boundary between sections A and B to the right, the character of the stream in section A changes from "all on" to "half on," or perhaps even to resemble a packet train. Hence, the specification of traffic primitives is contingent on the frame of reference, or the size of the "window" the observer looks through to see a portion of the traffic stream.

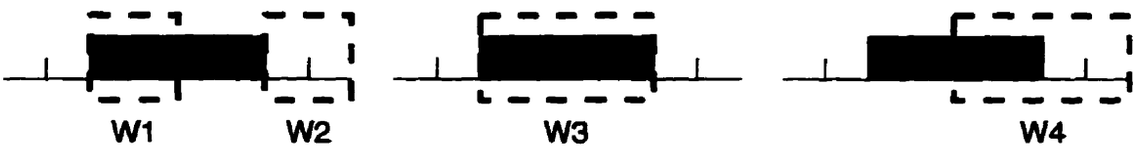
The smallest objects of a traffic stream, in the context of traffic primitives, are the cell and the slot. If the window through which an observer examines a traffic stream was only one slot large, then the observer could only say that a cell or slot is present; no other information about the stream could be inferred. As the window through which the observer sees the traffic stream is enlarged, more stream characteristics become apparent. In fact, the number of distinctions in traffic character that can be made when utilizing a window of size W is 2^W. To justify this, consider a window of size $W = 1$. The observer will see either a cell or a slot, and nothing else. If the window is of size $W = 2$, the observer will see either two cells, two slots, a cell and a slot, or a slot and a cell, as show in Figure 2.2. Representing a slot with a 0 and a cell with a 1, as in Section 1.6.4,

Figure 2.2: Possible Observations from a Window of Size $W = 2$

these distinctions can be expressed as the set {11, 00, 10, 01}. For a window sized $W = 3$, the set of distinct traffic patterns that can be observed through the window becomes {000, 001, 010, 011, 100, 101, 110, 111}. Accordingly, one can clearly see the reasoning for 2^W through elementary set theory: the number of distinctions that can be observed through a window of size W is equivalent to the number of elements in the power set[2] of a set of W elements. In this case, each element is a traffic stream position which can be observed through the window, and in which either a cell or a slot can be placed. This property of increasing distinctiveness as the observation window size increases is analogous to statistical methods, in that the more observations of a traffic stream are made, the more accurate become the statistical measures of the stream. At the extreme, the window could encompass the entire stream, at which point the observer would have complete knowledge of the source. However, performing this is as impractical when employing traffic primitives as it is when employing statistical measures.

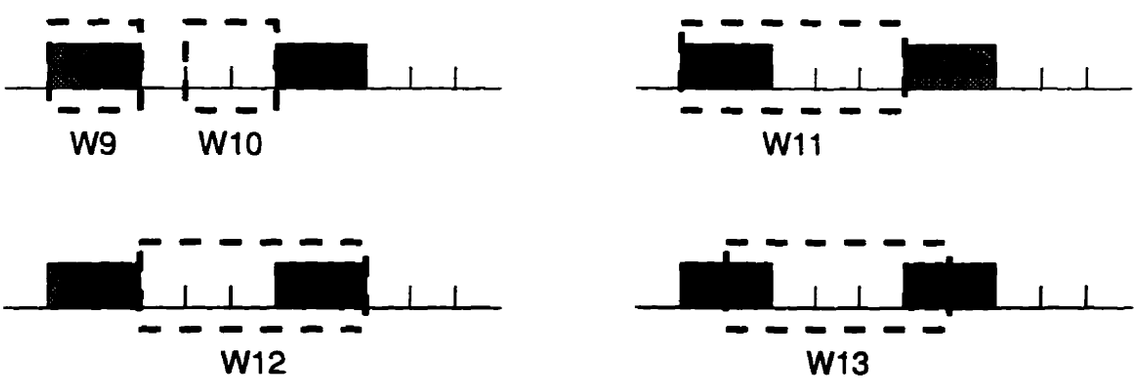
In order to show more clearly the effect of different window sizes and their relative positions on the traffic characteristics observed in the given traffic stream, sections A, B and C of Figure 2.1 are reproduced in Figures 2.3(a)–2.3(c), respectively. Consider section A of Figure 2.1, which is reproduced three times in Figure 2.3(a). The dashed boxes represent four possible sizes and positions of the window through which the observer views the traffic stream. As discussed above, a window of size one slot does not produce very interesting information, so the first window of Figure 2.3(a), W1, is two slots wide. With the amount of traffic stream observed

[2]For the definition of the power set, see [Gil76, 2.01 Definition], and for the number of elements in a power set, see [Gil76, 2.06 Theorem].



(a) Reproductions of section A

(b) Reproductions of section B



(c) Reproductions of section C

Figure 2.3: Effect of Window Size and Position on Stream Identification

through window W1, the observer may conclude that the transmission link is fully utilized, since the portion of the stream seen through this window contains only cells. For two cell arrivals, or as the window W1 is slid cell by cell towards window position W2, the observer could conclude the same. As the window is slid once more

to the right to include the first slot, the observer may wish to change this observation to report that only half of the link is being utilized. At W2, the observation could now be that the link is idle. These quickly changing observations are due to the fact that the observation window is quite small. In order to make fewer, more stable observations, and to increase the number of traffic characteristics which can be observed, window W3 is twice the size of W1. As can be seen, it encompasses the entire "on" period of this section of the traffic stream. The observer could now state with more confidence that the stream utilizes the full link rate, since more of the stream is being observed. However, as window W3 is slid to the right towards the position of window W4, the observer may wish to change this statement. At position W4, it now appears that the link capacity is only 50% utilized. Even with only one shift to the right, that is after only one additional observation, the observer may start to doubt the initial characterization of the stream.

Figure 2.3(b) reproduces, once again three times, section B of Figure 2.1, the section of the traffic stream which has the characteristics of a constant bit rate source utilizing 50% of the link capacity. As in Figure 2.3(a), using a window size of $W = 2$ and observing the traffic stream through window positions starting at W5 and sliding towards W6 on a cell by cell or slot by slot basis, it can be seen that the observer could make the characterization that the traffic stream is made up of a cell and a slot, in an alternating fashion. Using the notation introduced in Section 1.6.6.4, call this traffic stream type CBR.(1). It would appear that for this type of traffic stream, a window size of $W = 2$ is sufficient to capture its characteristics. However, this window size does not fare as well when used to characterize the

stream shown in Figure 2.4, which differs from that of Figure 2.3(b) by only one slot between each cell pair; call this traffic type CBR(2). With this stream, window positions X1 and X3 characterize the stream as type CBR(1), which it is not, and position X2 indicates that the stream does not utilize any of the link capacity, which it clearly does. Therefore, while a window size $W = 2$ can be used to capture the features of the constant bit rate stream of Figure 2.3(b) that utilizes half of the link capacity, it cannot be used to characterize the constant bit rate stream of Figure 2.4 which utilizes one third of the link capacity. As was suggested above when discussing Figure 2.3(a), larger window sizes are required. Referring back to Figure 2.3(b), window W7 is now six slots long. In addition to being able to characterize

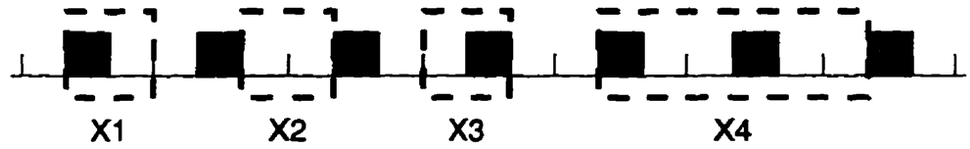


Figure 2.4: The Requirement for larger Window Sizes

the CBR(1) stream, this window can also be used to characterize the traffic stream of Figure 2.4 (window X4) as CBR(2), not CBR(1). This is another example of how increasing window size can be used to increase the distinctions that can be observed, and thus the number of traffic stream types. Unfortunately, as can be seen from window position W8 of Figure 2.3(b), it is still possible for the observer to question the characterization. In this case, the observer may wish to say that the stream is "close" to that of stream type CBR(1), since it differs by only one slot. This idea will be revisited in later sections.

As the final example of the effect of window size and position in observing characteristics or features in a traffic stream, Figure 2.3(c) reproduces, four times, section C of Figure 2.1. As can be seen, section C resembles a packet train, with two contiguous cells followed by three contiguous slots. To help this discussion, call this traffic type PT(2, 3), again using the notation of Section 1.6.6.4. As can be seen from windows W9 and W10 of Figure 2.3(c), a window size of $W = 2$ does not correctly capture the features of this stream. The observer may report that the traffic stream utilizes all of the link capacity, or none, or that the stream is type CBR(1). In order to capture the characteristics of this traffic stream, the window size must be increased. Accordingly, windows W11, W12 and W13 are five slots long. Window W11 obviously allows the observation of the features of the PT(2, 3) stream: two cells followed by three slots, the "definition" of this traffic type. However, it should be noted that the cell and slot patterns observed through windows W12 and W13 *also* show the features of this traffic type; the window W11 has simply been shifted along the traffic stream. This is an extremely important point to consider when deciding which traffic types are being observed: *not only the cell and slot pattern of the definition of a traffic stream type must be considered, but also any other patterns that are produced when shifting the window along the defining traffic stream.* The reason for this "rule" is for stability in making observations; that is, if the traffic stream observed does not change in character, then neither should the traffic stream type. If the observer is presented a traffic stream which is a true PT(2, 3) stream, then for all window positions and thus all observations of this stream, the observer should report that the stream is type PT(2, 3). If, on

the other hand, the observer sees the cell and slot pattern of, say, window W13 of Figure 2.3(c), and reports that this is traffic type, say, "XJS−12," then the definition of a pure PT (2, 3) stream, based on its traffic patterns, becomes unduly complicated. In other words, it would be possible *not* to follow the above rule, but it would only obfuscate the determination of the traffic stream type. Summing up, some features of certain traffic streams may have more than one cell and slot pattern, or *morphism*, as will be discussed in Section 2.2.1.

Bringing the ideas of this section together, it can be seen that as the window size W used to observe a traffic stream increases, so does the number of features or characteristics that can be picked out of the stream. The window can be thought of as a low pass filter, where the larger the window is, the more high frequency features are "filtered out." Since more of the stream is observed through a larger window, a single statement pertaining to the traffic stream's long-term characteristics can replace more frequent statements regarding the stream's short-term features. That is, a larger W decreases the chances of observing a stream with characteristics, say, that it utilizes the full link capacity, then half, then none, then back to half, then back to full, and so on, with the single observation that the stream is of traffic type, say, CBR (2). The choice of window size also impacts on the definition of the traffic stream types that can be used to specify the observations. A small window size limits the number of stream types which can be defined to 2^W. For example, with a window size $W = 5$, only $2^5 = 32$ different traffic characteristics can be differentiated. While this may be sufficient for determining whether a traffic stream is, say, CBR (2), it is not possible to define a sufficient number of features for

making more complicated classifications, such as the CBR (2) stream has become a PT (2, 3) stream. Making observations such as this, of course, is a useful application of traffic classification.

Also, many of the cell and slot patterns, or traffic patterns, observed through the window must be associated with the same traffic type in order to ensure stability in the observations made. These points tend to recommend that the window be made as large as possible, and theoretically, this would be ideal. On the other hand, too large a window can allow too much information about the traffic stream to be observed, thus making stream characterization difficult. In addition, the larger the window is, the more time will be required in making an informed decision about traffic type, since more data must be collected. The size of window chosen, then, represents a tradeoff between accuracy of observation and delay in making that observation. As will be seen in Section 2.1.4, practical problems that arise when training neural networks also have a limiting effect on the window size.

2.1.2 Partitioning of Traffic Streams

The discussion of Section 2.1.1 introduced the concept of the traffic primitive. Traffic primitives are based on features or characteristics observed in a given traffic stream. In addition, their specification depends on the size of the window through which the stream is observed. This section will serve as a link between the characterization of traffic streams and the formal specification of traffic primitives which will appear in Section 2.1.3. The determination of the window size W will be postponed until Section 2.1.4.

It is beneficial for the specification of the traffic primitives in an ATM setting to attempt to partition, or broadly pre-classify from the set of all possible cell and slot patterns Ψ, the traffic streams anticipated in the network into classes of traffic types. The partitions defined should have general properties of the streams contained therein. This partitioning, however, should not be so specific as to preclude the classification of new traffic types as they should arise — most likely from completely new applications.[3] For more specific properties of the streams in a given partition, traffic primitives must be defined. Nonetheless, as much as possible a priori knowledge of the system at hand should be employed in order to make the best partitioning decisions. In a general high speed network setting, an obvious choice for partitioning traffic streams is by the traffic types forseen to be prevalent. The drawback of this, of course, is the problem of new traffic types, as alluded to above. This problem may be avoided in an ATM network, however, due to its slotted nature, which tends to "normalize" the features of traffic streams. In an ATM network, before a data stream enters the backbone network, it passes through the ATM Adaptation Layer, introduced in Section 1.6.4. As discussed, regardless of the nature of the traffic stream entering the AAL, it will exit as groups of cells (bursts) and groups of slots (idle periods) [LTWW94]. Only rarely will a single cell or slot appear, except of course as an integral part of a given traffic type, such as a constant bit rate stream. Considering this, the contrived source shown in Figure 2.1 of the previous section foreshadows the traffic classes which will result after partitioning.

---

[3]For example, the use of the Internet (a data network) for voice applications was most likely completely unanticipated by the Internet's designers.

To specify a partition, it is easiest to consider the cell patterns that can be present after the AAL. Starting with simple patterns and increasing their complexity, there could be: all cells; all slots; a cell followed by a slot, repeating; a cell followed by two slots, repeating; a cell followed by N slots, repeating; two cells followed by a slot, repeating; two cells followed by two slots, repeating; two cells followed by N slots, repeating; and in general, M cells followed by N slots, repeating. Thus, by simply fixing the number of cells and slots a broad partition can be made of the features of traffic streams emitted by the AAL. Call this the *deterministic partition.* Another partition can be made along similar lines, where instead of a constant number of M cells are followed by N slots, repeating, the number of cells and slots are given by probability distributions. Since an ATM network is slotted and thus cell arrivals occur only at discrete times, this partition would contain traffic streams specified by discrete distributions. This may include, but is not limited to, Interrupted Bernoulli Processes (IBP), packet trains with on and off periods specified by, say, uniform or geometric distributions, and Markov-modulated Bernoulli processes (MMBP), as cited in Section 1.3.1. Call this the *distribution partition.* Specifying these two partitions encompasses many common traffic streams. A final partition should be specified which contains extremely rare sources, or more precisely cell and slot patterns which are sporadic or occur as a result of other (unanticipated) events. For example, a stream which contains only slots except for a single cell, or two cells followed by three slots followed by three cells, *not* repeating. Another group of streams would be those memoryless streams which can be represented by a geometric distribution; fortuitously, due to

the action of the AAL, streams with this characteristic should be extremely rare. Call this the *degenerate partition*. It could be argued that a fourth partition should be defined which contains all traffic streams not yet defined. However, it is hoped that by judiciously partioning known traffic streams and thus by specifying the traffic primitives properly, new traffic streams will consist only of traffic primitives already specified. In this way, the problems associated with previously unknown traffic streams will be circumvented.

It must be understood that these broad partitions are made as a consequence of the foreknowledge that all streams must pass through the AAL before being admitted to the backbone of an ATM network. In high speed networks other than ATM, the partitioning may be completely different, and hence the resulting specification of the traffic primitives. In these cases the traffic classification may yield different results, yet the method employed here can remain the same.

The number of traffic streams encompassed by each partition may be very large, and most likely countably infinite. It is impossible to train a neural network to learn an infinite number of training patterns, however, since this will take an infinite amount of training time. In addition, attempting to train a neural network with "random" inputs will also be difficult, since neural networks learn input-output relationships. If the input-output relationships are not static, then on each training pass the neural network will attempt to remember the new relationship. Thus, the neural network training patterns are chosen only from the deterministic and degenerate partitions. The guiding idea here is that the neural network will generalize from the deterministic patterns and be able to classify traffic streams from the

distribution partition. In fact, it is planned that variations from the deterministic partition traffic streams will be interpreted as "noise," and thus generalization will take place, as discussed in Section 1.5. For example, a packet train source with uniformly distributed "on" period of mean two cells and uniformly distributed "off" period of mean three slots should be classified, with a high degree of probability, as a PT $(2, 3)$ (deterministic) traffic stream. However, it is anticipated that some of the classifications of this "noisy" PT $(2, 3)$ source will also result in the observation of traffic types such as PT $(3, 3)$, PT $(4, 2)$, PT $(5, 1)$, for example. These traffic types are very similar to PT $(2, 3)$, and so the randomness in cell arrivals of this uniformly distributed source may lead to observations of a variety of traffic types. From this one may conclude that this stream will be classified poorly. This would be the case if only these initial classifications were used. For distribution partition traffic streams, however, further classification is required, where patterns in the traffic primitives must be observed. As mentioned previously, the classification of traffic streams is highly dependent on the classifications required. That is, if a network provider knows that only deterministic partition traffic streams will be present in a certain situation, then only an initial classification would be required. This will be discussed further in Section 2.1.6 and Chapter 3.

As a final note, the degenerate partition is included for those few sources that may require a special traffic class, for example all slots (an idle link). With the partitioning of traffic streams completed, it is now possible to characterize individual traffic classes within the partitions, and thus specify the traffic primitives.

2.1.3 Specification of the Traffic Primitives

The specification of the traffic primitives is dependent not only on the traffic stream partitions, but also the size of the training window. As a backdrop to this, and perhaps more importantly, the traffic primitive specification is intimately tied to the desires of the network service provider. As mentioned earlier, the network provider most likely wishes to maximize revenue, which corresponds to utilizing the network bandwidth in the most efficient manner. Thus the traffic primitives should be chosen to achieve this goal. Then, by way of some control mechanism. one method to maximize efficiency could be to simply detect what traffic primitive is present. Another method could be to assume that a given traffic stream is present, and watch for deviations from this type. These applications of primitive classification will be discussed further in Chapter 3. In any case, it is important to realize that the desired outcome of primitive classification has a great impact in the specification of these same traffic primitives.

Working with this in mind, the traffic primitives specified here are chosen with some trepidation. Instead of tying the specification to the desires of some imaginary network service provider where assumptions must be made about the provider's goals, the traffic primitives were chosen to showcase their usefulness, applicability to an ATM setting, and scalability. As a result, if one questions the validity of a certain traffic class, it should be understood that the class can easily be changed to suit other traffic streams or situations, or at least other traffic classes can be defined. Also note that the size of the traffic window W is kept variable in this section, and will be determined in Section 2.1.4. While the traffic primitives specified depend

on W, this dependency is in a relative sense, not absolute.

The traffic primitives chosen are Constant Bit Rate (CBR), CBR with Rate Changes (CBR-RC) and Deterministic Packet Train (PT) from the deterministic partition, and Degenerate (DG) from the degenerate partition. The following sections specify the *base primitives* for these four primitive classes.[4] As the rule in Section 2.1.1 alluded to, most traffic primitives have more than one morphism, based upon the length of the primitive itself and the window size W. Since the morphisms are not important to the specification of the (base) traffic primitives, and since the question of window size is answered in the next section, only the base traffic primitives are discussed here. In fact, the traffic primitive morphisms require consideration only during neural network training, and so their specification is delayed until Section 2.2.

2.1.3.1 CBR (Constant Bit Rate)

CBR traffic primitives are those which result from the cell and slot pattern of a traffic stream that has a rate which is a constant integer fraction of the link rate C, and are still unique. That is $\frac{C}{i}$, where $1 \leq i \leq W-1$ (for $i \geq W$, see Section 2.1.3.4). This definition allows base primitives corresponding to a traffic stream rate of C, $\frac{C}{2}$, $\frac{C}{3}$, ..., $\frac{C}{W-1}$. This primitive class is chosen so that CBR traffic streams can be recognized directly, as discussed in Section 2.1.1, Figure 2.3(b). It should capture most traffic streams with the characteristic of a single cell followed by N slots, repeating. In addition, as an example of how different traffic streams

[4]The reader may wish to refer to Figures 2.9–2.12 of Section 2.2.1, which depicts these base primitives as well as their morphisms for a window size of $W = 10$.

can be detected as a single class, these traffic primitives do not specify the rate of the traffic stream. That is, all CBR traffic streams of rate $\frac{C}{i}$, $1 \leq i \leq W-1$, or CBR(0) through CBR$(W-2)$ (using the notation of Section 1.6.6) are classified simply as a CBR stream. As mentioned above, however, a network provider would most likely wish to have rate information. To reiterate, the traffic primitives here are chosen not for a specific ATM network situation, rather to show the utility of neural network based traffic classification.

2.1.3.2 CBR-RC (CBR with Rate Changes)

These traffic primitives are specified from the characteristics which result when a traffic stream changes rate from one constant integer fraction of the link rate C to another; that is $\frac{C}{i} \rightarrow \frac{C}{j}$, $i \neq j$, where $1 \leq i, j \leq W-1$. This definition corresponds to a source CBR(0) through CBR$(W-2)$ changing to a source CBR(0) through CBR$(W-2)$, and of course realizing that a change from CBR(i) to CBR(i) is no change at all. This traffic primitive class is chosen with a specific application of traffic classification in mind, namely the detection of a CBR traffic stream which has broken its traffic contract. Again, as previously stated, while this class may not be useful to a specific network provider, it does show how traffic primitives can be employed.

2.1.3.3 PT (Deterministic Packet Train)

The PT traffic primitives make up the majority of the neural network training vectors that are a consequence of traffic primitive specification. They characterize the packet train streams of the deterministic partition, and should also be useful

for generalizing to many of the traffic streams in the distribution partition. The base traffic primitives result when two or more cells are followed by one or more slots. This corresponds to the sources $\mathrm{PT}(2,1)$ through $\mathrm{PT}(2,W-2)$ through $\mathrm{PT}(W-1,W-2)$ through $\mathrm{PT}(W-1,1)$. Since the AAL is expected to produce streams that fall into this primitive class, this class is integral to good traffic classification. Again, as in the CBR case, all of the different packet train base primitives, and their morphisms, will result in the same traffic classification. Hence, a traffic stream which is of type $\mathrm{PT}(2,3)$ and one which is $\mathrm{PT}(3,5)$ will result in the same traffic primitive, namely PT. If an application of traffic classification requires greater resolution, then more specific packet train classes can be defined.

2.1.3.4 DG (Degenerate)

These traffic primitives could belong to either of the CBR, CBR-RC or PT primitive classes, and are placed in this class to keep the other primitives unique.[5] For example, traffic streams with rates $\frac{C}{W}$ or less, that is $\frac{C}{k}$, $k \geq W$, are indistinguishable when viewed from a window of size W; when observed, these streams appear as either all slots, or a single cell and all slots. In general, this primitive class represents sources $\mathrm{CBR}(W-1)$ through $\mathrm{CBR}(\infty)$. This class also contains unique occurrences, such as the primitive which represents an idle link. Note however, that a window completely filled with cells is equivalent to a transmission rate of $\frac{C}{1}$, a $\mathrm{CBR}(0)$ source, which is *defined* to be a CBR primitive. Again, this is somewhat

[5]Neural network training vectors must be unique in the following sense: a neural network can learn a many to one input-output mapping, but not a one to many. Thus, for any given input to the neural network, only one output is possible.

arbitrary, and could be changed to be a PT primitive, for example.

2.1.4 The Primitive Classifier

With the traffic primitives defined, the *primitive classifier* is introduced in this section. In addition, the size of the "traffic window," W, that is the size of the window through which the traffic stream is observed by the neural network, is also determined. Figure 2.5 depicts a cellized traffic stream as it is presented to the

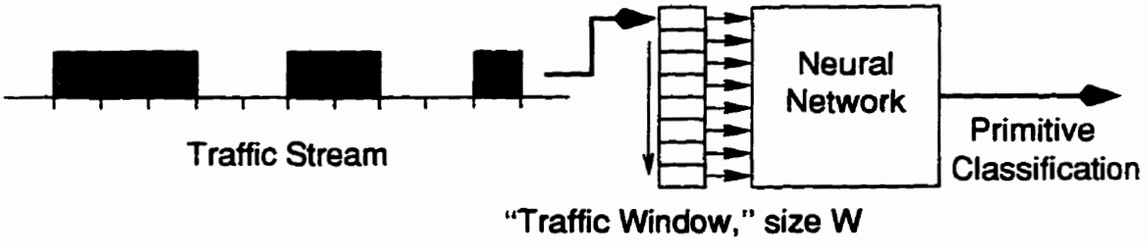


Figure 2.5: The Traffic Primitive Classifier

primitive classifier, which then outputs the traffic primitive recognized. As shown, the engine of the classifier is a neural network. As a traffic stream is presented to the network access point, it undergoes cellization at the AAL. This is represented by the traffic stream in Figure 2.5. Each traffic window of cellized traffic stream presented to the neural network causes a classification to be made which results in a traffic primitive at the neural network's output. As a new cell or slot exits the AAL, the traffic window is updated by dropping the oldest cell or slot observation and adding the newest, in a shift-register fashion. If the cellized traffic stream is represented by a string of 1's and 0's, as discussed in Section 1.6.4, then the

input to the neural network is a window consisting of 1's and 0's. The fact that the inputs are discrete reduces the neural network training time. As can be seen, the neural network based primitive classifier is very simple. Contrast this to a conventional (statistical) control scheme, which would require the calculation of statistical measures taken on the traffic stream, and then a complicated control algorithm, as mentioned in Section 1.3. Of course, it must be noted that much of the complexity is now within the neural network.

The determination of the size of the traffic window, W, is not as simple. Much has already been said about the the effect of the window size on classification in Sections 2.1.1 and 2.1.3. To recap, the salient point is that the traffic window size represents a tradeoff between accuracy of classification and reaction time of the primitive classifier based control mechanism. On the one hand, the smaller the window size the faster the reaction to changes in the traffic stream, as well as the smaller the initial delay in filling the traffic window. On the other hand, the smaller the window size, the fewer the number of traffic primitives that can be defined, which is limited to 2^W.

If these are the only considerations, then the ultimate choice of W is determined by the traffic streams a network provider plans to classify and thus allow into the network. For streams which change character more frequently, larger window sizes are warranted. This results in greater observation stability at the cost of more complicated traffic primitives and classes. For streams which do not change character often, then smaller windows can be employed, since fewer traffic distinctions are present. A window sized $W = 100$ represents about 0.24 ms of a source transmit-

ting at $C_L = 155$ Mbps. For an MPEG[6] traffic stream with mean rate 1.5 Mbps [LeG91], the same window size represents about 0.02 ms. Thus, for "lower" rate traffic streams, or additionally for delay insensitive traffic streams, large window sizes are required.

However, while the tradeoff between the two factors above is important, it is moot in light of the greater consideration when dealing with neural networks: training time. A larger window size implies that a greater number of more complicated traffic primitives need to be learned by the neural network. This necessitates larger, more complicated neural networks, which in turn means more presentations of the training vectors will most likely be required before training is successful. Thus, while a traffic window sized $W = 100$ may be usable when considering the time scale of traffic streams to be classified, it may be prohibitive when considering the training time required. This is the case here. Due to the limits imposed by the computing power available, the largest window size that was trainable in this work is $W = 20$. This window size represents about 47 μs of a source using the entire link rate, and about 4.9 ms or $\frac{1}{6}$ of a frame of the MPEG source mentioned above. The details of the neural network training, including training times are discussed in Section 2.2. Before this, though, the following section describes traffic primitives with an analogy to optical character recognition.

[6]Motion Pictures Expert Group.

2.1.5 Traffic Primitives as "Traffic Characters"

It may be interesting for the reader familiar with Optical Character Recognition (OCR) [Guy91, LBD+89, Pao89], one of the most well known uses for neural networks in general and Backpropagation in particular, to consider the following analogy. Traffic primitives can be thought of as "traffic characters" to be recognized by an OCR system. In such, the neural network is presented with the character to be recognized, as well as any allowable "morphisms" of that character, such as shifts and rotations. Once the neural network learns all the characters to be recognized, as well as their morphisms, it can then be expected to correctly recognize and then classify characters which it learned — even in the presence of noise. The left hand side of Figure 2.6 shows an example of this for the character "T." The top box depicts the base character, superimposed on a grid of 100 pixels which are either "on" (grey) or "off" (white). The middle grid shows the "T" character shifted one pixel to the right; this is but one of many possible morphisms of the base character. The bottom grid gives an example of some noise on the pixel grid. After successful training, all three of the pixel grids on the left side of Figure 2.6 produce the same output, namely a "T" is present at the input of the neural network.

Similarly, the right hand side of Figure 2.6 shows an example of a possible traffic primitive. For illustrative purposes only, a packet train traffic stream consisting of two cells followed by three slots, repeating, is depicted; that is PT $(2, 3)$ as discussed in Section 2.1.1. While in the OCR case the underlying grid is made up of pixels, for the traffic primitive case the grid is made up of (ATM) cells and slots. If there is a cell present, then the grid square is grey, whereas if a slot is present, the grid square

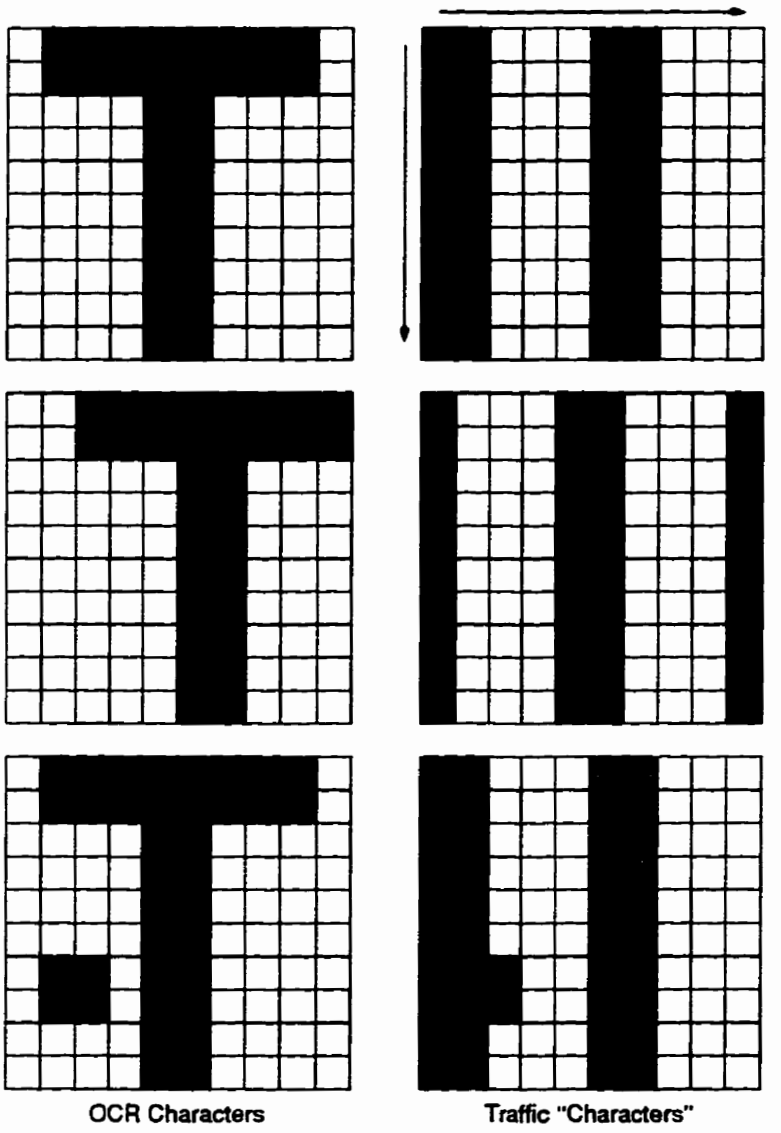


Figure 2.6: Traffic Primitives as OCR Objects

is white. Also note that the slots are placed on the grid from left to right and from top to bottom, as the arrows on the figure indicate. Hence, the top grid on the right

hand side of Figure 2.6 defines the 100 cell and slot[7] version "base character" of this PT $(2, 3)$ base traffic primitive. One possible morphism of this primitive is given in the middle right hand grid of the figure, which is the base primitive "shifted" to the right by one cell. After some inspection the reader should realize that this traffic primitive has three other *unique* morphisms. Also realize that this 100 cell and slot grid is equivalent to a traffic window, as discussed previously, of size $W = 100$. The window has simply been rearranged in order to show traffic classification in a manner comparable to OCR. Finally, the bottom grid shows the traffic primitive in the presence of "noise." To complete the analogy, the traffic characters on the right hand side of Figure 2.6, after successful training, will produce the same output, namely a PT $(2, 3)$ traffic stream is present at the input of the neural network.

The fact that neural networks can operate in the presence of noise is well known, and indeed this ability plays a prominent role in the concept of this research, that is the treatment of traffic streams as strings of objects rather than in a statistical sense. From an ATM network control point of view, it is undesirable for the identification method of traffic streams to be too sensitive to a few cells being "out of place," or to be so insensitive that two dissimilar streams are classified as being identical. Thus, if the traffic stream changes character for very short durations, these fluctuations can be thought of as a noise component on the "stationary" traffic stream. Hence. the neural network primitive classification, which is inherently robust in the presence of noise, will not change. This creates a level of sensitivity and stability which may not be attainable by statistical methods.

---

[7]Read pixel.

2.1.6 Scalability of the Classifier: The Compound Classifier

With the appropriate choice of traffic primitives, a great many traffic streams are classifiable. Others, however, may require using two or more primitives in a certain order to identify the stream, which would define a *traffic compound.* While it could be argued that traffic compounds are unnecessary since increasing the size of W allows the definition of more primitives, due to the exponential nature of neural network training time with the number inputs and thus weights in the neural network, it is better to train two networks with N weights than one network with $2N$ weights. Thus a device such as that of Figure 2.7 can be employed. The traffic compound classifier operates in much the same way as the primitive classifier of

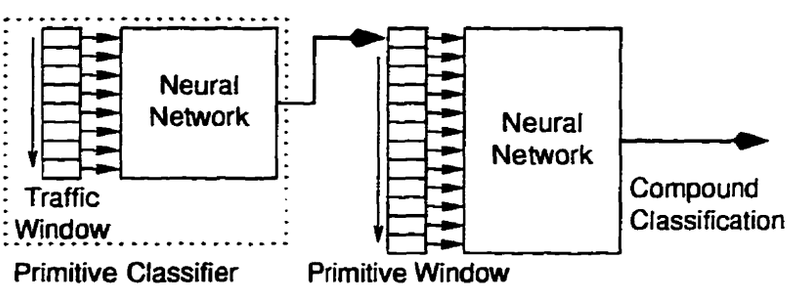


Figure 2.7: The Traffic Compound Classifier

Figure 2.5, but instead of observing the traffic stream directly, it observes a sequence of traffic primitives. Its job is to look for patterns or identifying features in the traffic primitives and further classify the traffic stream into traffic classes which encompass more complicated streams. In addition, using a series of compound classifiers, certain non-stationary features of a traffic stream can be detected or

special situations can be identified.

For the neural network purist, it is noted that the traffic primitive and compound classifier can be combined to form one neural network of approximately the same size. This is not attempted here, however, since it obscures the basic idea of this work. In addition, this combination is highly dependent on the traffic streams to be classified and the intended use of the classifications, as discussed in Section 3. Therefore, since the training and operation of the compound classifier is very similar to that of the primitive classifier—with the traffic stream replaced by a sequence of traffic primitives and the traffic primitive classifications replaced with traffic compound classifications—the compound classifier is not implemented.

2.2 Neural Network Training

A basic, vanilla-flavored, Backpropagation training algorithm [Hay94, RHW86], as derived in Appendix A, is implemented on a fully connected neural network, as shown in Figure 2.8. The neural network training and simulation software suite is all original, first written in C and then ported to C++. The choice of the

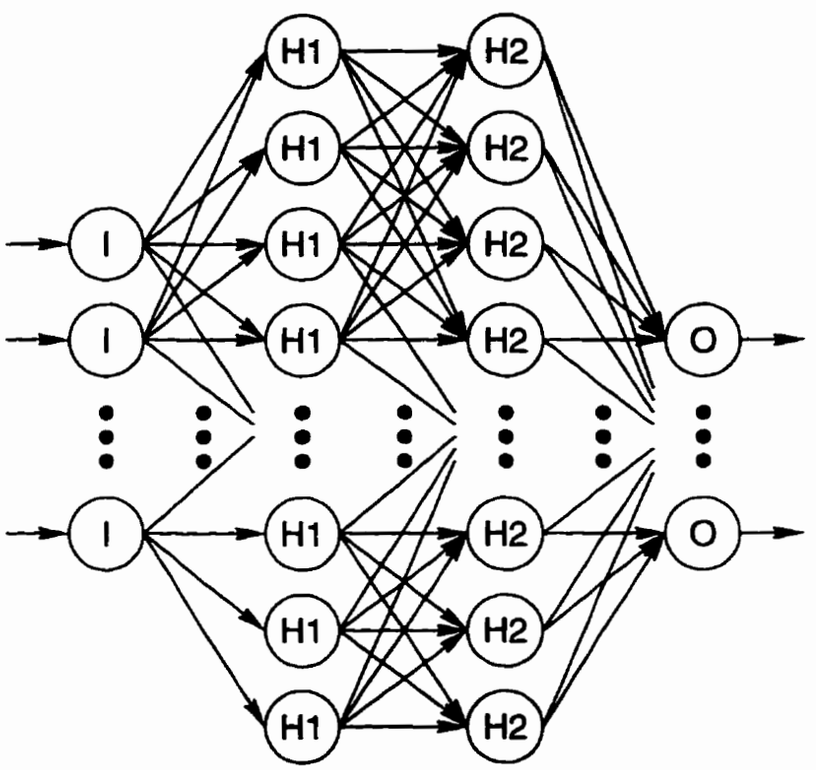


Figure 2.8: Neural Network Topology

neural network topology is very problem specific, and much of the literature deals with just this problem. As stated, however, it is not the intention of this work to make contributions in the field of neural network training. Thus, this simple

fully-connected network is chosen.

Generally, the layout of the neural network to be trained is very much determined by the number of inputs and outputs desired, which in itself is determined by the problem at hand. With the case of primitive classification, the number of inputs to the neural network is determined by the size of the traffic window, W, of the primitive classifier of Figure 2.5. As mentioned in Section 2.1.4, the choice of W is highly dependent on the traffic streams to be classified and the type of classification to be performed. Recall, though, that the time required to train the neural network must also be taken into consideration.

Therefore $W = 10$ is chosen, since it allows the definition of a fairly large number of traffic primitives from the observable stream characteristics, namely $2^W = 1024$ as discussed in Section 2.1.1, but more importantly since it allows the training of the neural network with the computational power available.[8] It is acknowledged, though, that such a small window size may not be large enough to capture the characteristics of complicated sources. In order to form points of comparison as to the effect of the traffic window size on traffic classification, two additional neural networks are trained, one with $W = 15$, and the other with $W = 20$. These allow the observation of $2^{15} = 32,768$ and $2^{20} = 1,048,576$ traffic stream characteristics, respectively, and so permit the definition of many more primitives. The training time of the neural network with twenty inputs, as will be seen, is enormous.

To determine the number of neural network outputs, each of the traffic primi-

[8]For the reader unfamiliar with training neural networks, it must be realized that a neural network cannot simply be trained once. Many training sessions must be performed in order to "tweak" the training parameters with respect to the situation at hand. Thus, the training of a single large neural network represents a vast investment in computation time.

tive classifications desired is given an unique binary code. For example, the CBR primitives of Section 2.1.3.1 represent one classification, as do the PT primitives of Section 2.1.3.3 and the DG primitives of Section 2.1.3.4. However, since it is the aim of the CBR-RC primitives to show that the classifier can detect changes in a traffic stream, the CBR-RC primitives of Section 2.1.3.2 represent many classifications. The number of binary codes required, then, is determined by generating the desired traffic primitives and observing the number of classifications they represent. This is performed via the software written, using an exhaustive search of all possible binary codes, given the window size W. In addtion, Section 2.2.1.2 describes how the number of binary codes can be estimated, and thus the number of neural network outputs. Summarizing the numbers here, the ten input neural network requires nine output neurons, and the fifteen and twenty input neural networks require ten and eleven outputs, respectfully. To complete the statement of the neural network topology, only the number of hidden layers and the number of neurons in each hidden layer are required.

The number of hidden layers and neurons, as is the usual method of the literature, is determined in a heuristic manner, and is summarized in Appendix B. First, small neural networks are trained, with one hidden layer and but a few neurons. When they fail to converge, the number of neurons and, or, hidden layers are increased until training convergence occurs. As more results are obtained, experience with the training problem is gained. This leads to the decision that two hidden layers are required by the neural networks, with thirty-five, eighty and two hundred neurons in each hidden layer of the ten, fifteen and twenty input neural networks,

respectively.

Using the notation developed in Section A.1.3 of Appendix A, then, the preceding discussion a 10–35–35–9, a 15–80–80–10 and a 20–200-200-11 neural network. The following section describes the training vectors employed to train these neural networks to perform traffic classification. In order to speed-up the convergence in training, Section 2.2.2 briefly describes some of the methods incorporated into the software written. Finally Section 2.2.3 presents the results of training, including training error and training times.

2.2.1 Specification of the Training Vectors

The number of traffic vectors required depends on the number of traffic primitives as defined in Section 2.1.3, which in turn is determined by the size of the traffic window, W. As discussed in the previous section, the number of traffic primitives also determines the number of distinct traffic classifications, and thus the output size of the neural network. For the cases of PT and DG primitives, determining the number of traffic primitives that result from a traffic window size W is relatively straightforward. However, to determine the number for the CBR and CBR-RC primitives, their morphisms must be considered, and also the preservation of the uniqueness of the training vectors.

As alluded to in Section 2.1.3.4, a neural network can perform a many to one mapping, but not a one to many. Thus, while it is possible to train a neural network to give the same output for two different inputs, it is not possible for a neural network to give two different outputs for one input. Hence, if the same traffic

primitive results in two different classifications, the neural network training will not converge. In other words, the traffic primitive classifications must be mutually exclusive.

The number of traffic primitives that can be defined is limited by the number of traffic characteristics that can be observed through a window of size W, which is 2^W as discussed. However, the number of traffic primitives defined should be much less. If not, as mentioned in the literature and Appendix A, the neural network will operate as a look-up table and fail to generalize, abrogating its major advantage.

Section 2.2.1.1 gives some examples of the primitives used to train the ten input neural network. At this point, the training vectors themselves should be presented. However, using the definitions of the primitives in Section 2.1.3, there are 435 training vectors for the 10–35–35–9 neural network, 2,004 for the 15–80–80–10 and 5,996 for the 20–200–200–11 neural networks, respectively, as shown in Table 2.1. In addition, the table shows the maximum possible observations, 2^W, and the percentage of this value that the total number of training vectors represents. While the calculation for the specific types of training vector shown in the table is discussed in Section 2.2.1.2, since specifying the training vectors for the fifteen and twenty

Table 2.1: Number and Type of Training Vector for the Three Neural Networks Trained

Neural Network	Training Vectors Per Primitive				Total	2^W	% of 2^W
	DG	CBR	PT	CBR-RC			
10–35–35–9	11	25	156	243	435	1,024	42.48
15–80–80–10	16	56	546	1,386	2,004	32,768	6.12
20–200–200–11	21	100	1,311	4,564	5,996	1,048,576	0.57

input neural networks—in a human-readable format—would require approximately two hundred pages, only the training vectors for the ten input neural network are given. The reader should rest assured that the training vectors not included are very similar to those of the ten input neural network, the only difference being the number of neural network inputs and outputs, and the fact that there are a lot more training vectors for each type of traffic primitive.

The 435 training vectors for the 10–35–35–9 neural network are tabulated in Appendix C. As can be seen, binary inputs are used since the primitives represent cell and slot patterns derived from cellization, as introduced in Section 1.6.4. As well, the desired output of the neural network for each input is also shown. The outputs correspond to the classifications designed. Referring to Table C.1 in Appendix C, the eleven DG training vectors represent a single classification. The twenty-five CBR training vectors of Table C.2 and the 156 training vectors of Table C.3 correspond to two more classifications, respectively. Finally, each of the 243 CBR-RC training vectors of Table C.4 represents a single classification. Thus, the total number of classifications of the 10–35–35–9 neural network is 246, which requires an eight digit binary code.[9,10] Since the CBR-RC traffic primitives specify when a traffic stream changes its CBR rate, as defined in Section 2.1.3.2, the fol-

---

[9]The number of digits required, ϑ, is given by $2^{\vartheta} = 246$, or $\vartheta = \frac{\ln(246)}{\ln(2)} \approx 7.943$, which requires eight digits.

[10]The reason the 10–35–35–9 neural network was chosen over the 10–35–35–*8* that is specified by the 246 classifications stems from the fact that this work is experimentally based. In order to experiment with the classifications, the number of neural network outputs chosen should allow each of the 435 training vectors to have its own classification. Thus the required digits is $\vartheta = \frac{\ln(435)}{\ln(2)} \approx 8.765$, which requires nine digits. This is also true for the fifteen and twenty input neural networks.

lowing notation is used. The traffic primitive RC $\frac{C}{i} \rightarrow \frac{C}{j}$ specifies that a stream has changed its rate from $\frac{C}{i}$ to $\frac{C}{j}$ of the link rate. Thus RC $\frac{C}{4} \rightarrow \frac{C}{2}$ specifies that the source in question has increased its transmission rate from one quarter to one half of the link rate.

The following section gives a few examples of the traffic primitives used to define the training vectors, and Section 2.2.1.2 discusses the calculation and estimation of the number of training vectors required for neural networks.

2.2.1.1 Example Training Primitives for the Ten Input Neural Network

Figures 2.9–2.12 present a few examples of the traffic primitives used to define the training vectors of the 10–35–35–9 neural network, and thus $W = 10$. It is important to keep the traffic window size in mind, since it limits the number of traffic primitives that need be defined. Figure 2.9 depicts the base traffic primitive and its morphisms for a CBR source transmitting at a rate $\frac{C}{3}$. A portion of this

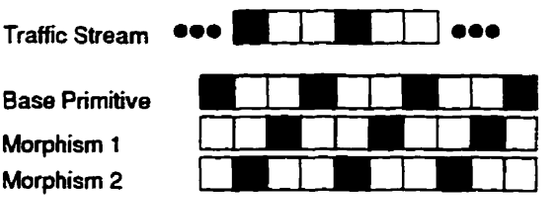


Figure 2.9: An Example CBR Primitive and its Morphisms

traffic stream is shown at the top of the figure. As can be seen, including the base primitive and morphisms, this definition gives rise to the three traffic primitives shown. The two morphisms are obtained by simply sliding the traffic window

towards the right along the traffic stream. Traffic primitives of this type can be used to detect constant bit rate sources CBR(0) through CBR(8), that is sources transmitting at the full link rate C to those transmitting at a rate of $\frac{C}{9}$.

Figure 2.10 shows an example of a PT base primitive definition, in particular a PT(2,3) source. Again, sliding the traffic window to the right four times specifies

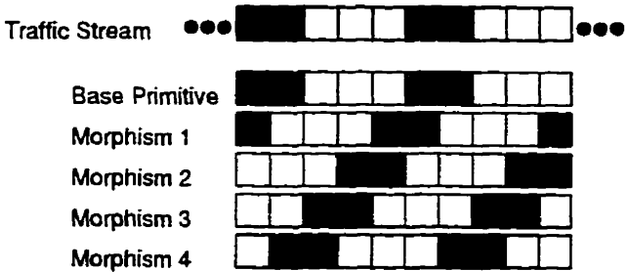


Figure 2.10: An Example PT Primitive and its Morphisms

the four traffic morphisms. Packet train traffic sources from from PT(2,1) through PT(2,8) through PT(9,8) through PT(9,1) can be detected from the definition of these primitives.

Figure 2.11 depicts the complete set of DG primitives for $W = 10$. These can be used to indicate that a low rate source is present, such as one that transmits at a rate of $\frac{C}{10}$, or less. In addition, a source which becomes idle can also be detected.

Finally, Figure 2.12 shows the CBR-RC base primitive and morphisms that arise when a CBR source changes its transmission rate from $\frac{C}{2}$ to $\frac{C}{3}$, that is from a CBR(1) source to a CBR(2) source, which corresponds to the RC $\frac{C}{2} \rightarrow \frac{C}{3}$ traffic primitives. As the figure attempts to show, in order to generate all the morphisms, one must consider transitions from the base primitive and morphisms of a CBR(1)

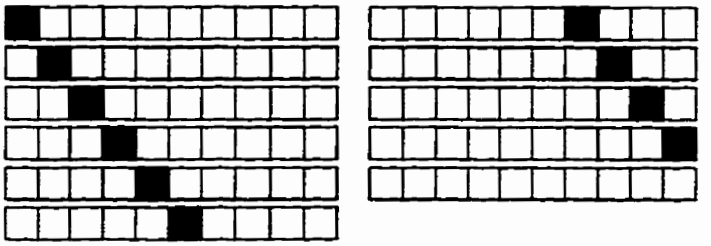


Figure 2.11: The DG Primitives for $W = 10$

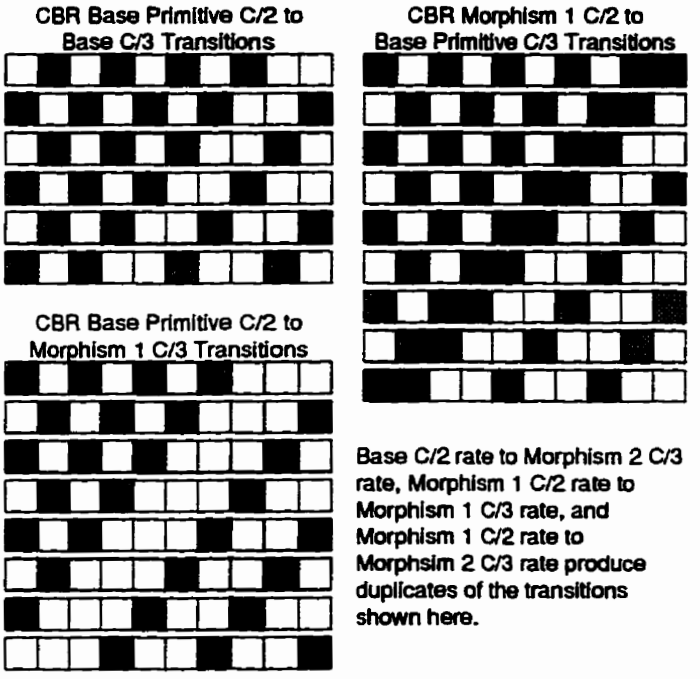


Figure 2.12: An Example CBR-RC Primitive and its Morphisms

source to the base primitive and morphisms of a CBR(2) source, including all possible window positions along the transition. Since there are two primitives specified by the CBR(1) source and three primitives specified by the CBR(2) source, and

since the transition of these two sources creates the need to consider $2W$ slots, there can be 120 morphisms of this primitive. Fortunately, most of these are repetitions of each other, as Figure 2.12 indicates.

As will be discussed in the following section, the number of RC $\frac{C}{2} \rightarrow \frac{C}{3}$ primitives can be reduced further, since the CBR-RC primitives are given the lowest "priority" when they are used to define the training vectors.

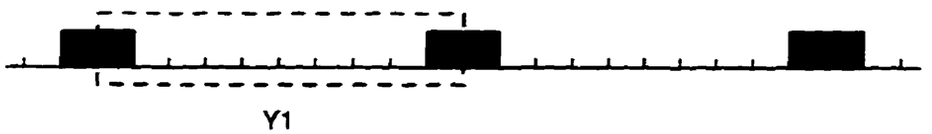
2.2.1.2 Traffic Primitives and Their Morphisms

The concept of a traffic primitive morphism has already been introduced in Section 2.1.1, when discussing the pros and cons of traffic window positions and sizes, and the "rule" stated. Simply, a traffic morphism results when a traffic stream with a constant set of features is observed from different points. Referring to Figure 2.3(c), windows W11, W12 and W13 all give different observations of the *same* traffic stream. Hence, these observations and the traffic primitives they define are termed morphisms. Since the training vectors are defined directly from traffic primitives and their morphisms, knowing the number of traffic primitives specifies the number of training vectors.

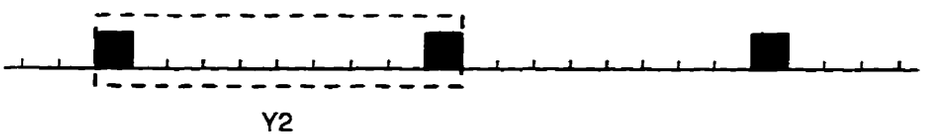
With the above definition, the reader may also note that since the DG, CBR and PT primitives each give a single traffic classification, they could also be considered to be morphisms. For example, ten of the DG training vectors of Table C.1 in Appendix C are defined from the ten morphisms of the cell and slot pattern of a single cell arrival when observed through a window of size $W = 10$ and shown in Figure 2.11. The eleventh DG training vector results from a window filled with all slots, which is part of the definition of the DG traffic primitives (see Section

2.1.3.4).

In addition, since the training vectors must be unique with respect to their classifications, as discussed in Section 2.2.1, then as training vectors are defined by the traffic primitives, they must be checked with the vectors previously specified in order to ensure that they result in mutually exclusive classifications. However, as discussed in Section 2.1.1, different traffic streams may appear to be the same, depending on the position and size of the traffic window. This leads to an interesting situation for the specification of the training vectors. Depending on the purpose of performing traffic classification that the network provider has in mind, priority should be given to the traffic primitives of the traffic class or classes that the network provider wishes to detect. In this context, priority refers to which set of traffic primitives is used to first define training vectors. For example, if the training vectors for PT traffic primitives are defined before any other, then the base primitives or morphisms of another class of traffic primitives, say CBR, which would cause an intersection of the PT and CBR classifications, would *not* define a training vector. This situation is shown in Figure 2.13. A packet train source with on-period of two cells and off-period of eight cells, or PT$(2, 8)$, is shown in Figure 2.13(a), and a constant bit rate source which utilizes one ninth of the link rate, or CBR(8), is shown in Figure 2.13(b). As can be seen, for a traffic window of size $W = 10$, the relative positions of window Y1 on the PT stream and Y2 on the CBR stream lead to the same traffic primitive. Thus, if the PT primitives are given priority over CBR primitives, the traffic primitive of window Y1 would give rise to a training vector whereas the primitive of window Y2 would not.



(a) Packet Train Stream, PT (2, 8)



(b) Constant Bit Rate Stream, CBR (8)

Figure 2.13: The Need for Priority amongst Traffic Primitives

For the training vectors chosen here, priority is given to DG, then CBR, then PT and finally CBR-RC. Hence, the potential training vector of any CBR-RC base primitive or morphism is checked against all the existing training vectors defined by the DG, CBR and PT primitives before it is added to the training set. In order not to be caught up with imaginary requirements of some fictitious network provider, the reason for this priority is simply that it follows from the actions of the AAL, described in Sections 1.6.4 and 2.1.2. The DG primitives are given the highest priority since they indicate streams which cannot be distinguished, the CBR follow simply because peak rate allocation is likely to be the first mode of operation of CAC in ATM networks, then PT primitives since they should be the most common, and thus the "default" source. The CBR-RC primitives, as mentioned, are defined with the CBR rate discrimination task in mind.

Once the traffic window size W is chosen, it is relatively easy to specify the number of DG and CBR traffic primitives that result from the definitions of Section 2.1.3. Since there is one DG primitive for each slot of the traffic window, plus the window completely filled with slots, the number of DG training vectors, N_{DG} that are defined by the DG traffic primitives is

$$N_{DG} = W + 1. \tag{2.1}$$

Therefore, for the ten, fifteen and twenty input neural networks trained, there are eleven, sixteen and twenty-one DG training vectors.

For the CBR primitives, if one shifts about a few CBR streams in a window, it can be seen that the midpoint of the window can be used to deduce the number of traffic primitives and thus training vectors. Define Υ as the smallest integer which divides the window size W in half, that is

$$\min_{\Upsilon \in \mathbb{Z}^+} \left(\left\lceil \frac{W}{\Upsilon} \right\rceil = 2 \right), \tag{2.2}$$

where $\lceil x \rceil$ denotes the ceiling of x, that is the smallest $k \in \mathbb{Z}^+$ such that $k \geq x$. Then the number of CBR training vectors, N_{CBR}, is given by

$$\sum_{i=1}^{\Upsilon} (i+1) \qquad \text{for } 1 \leq i \leq \Upsilon, \tag{2.3}$$

$$\sum_{i=\Upsilon+1}^{W} (W-i-1) \qquad \text{for } \Upsilon < i \leq W, \tag{2.4}$$

where i should be considered to be a dummy variable. Bringing these two cases

together yields

$$N_{CBR} = \frac{1}{2}\left[W^2 + 2\Upsilon^2 - 2W\Upsilon - 3W + 6\Upsilon - 4\right]. \tag{2.5}$$

A special case occurs if $\Upsilon = \frac{W}{2}$. Then $N_{CBR} = \frac{W^2}{4}$. Thus, for the ten, fifteen and twenty input neural networks, there are twenty-five, fifty-six and one hundred CBR training vectors.

Attempting to calculate the number of training vectors for PT and CBR-RC primitives soon breaks down, due to the number of non-unique base primitives and morphisms generated. For example, the total number CBR primitives given by the definition of Section 2.1.3.3 is

$$\sum_{i=2}^{W-1}\sum_{j=1}^{W-i}(i+j) = \frac{1}{6}\left[2W^3 - 3W^2 - 5W + 6\right], \tag{2.6}$$

but this would lead to many training vectors which would cause intersecting classifications. It is discovered from experience that Equation (2.6) overestimates the number of PT training vectors by a factor of almost two, and the larger the window size the smaller the error. Thus, the total number of PT training vectors, N_{PT}, for a traffic window of size W is

$$N_{PT} \lessapprox \frac{1}{12}\left[2W^3 - 3W^2 - 5W + 6\right]. \tag{2.7}$$

This approximation is compared with the actual number of training vectors required for various sizes of W in Table 2.2, showing the factor of underestimation. Since

Table 2.2: Estimating N_{PT} for Various Values of W

Window Size W	N_{PT}		Estimate Factor
	Actual	Estimate	
10	156	138	0.885
15	546	501	0.917
20	1,311	1,226	0.935
25	2,576	2,438	0.946
30	4,466	4,263	0.955

analytical methods for determining the number of training vectors required are not easily devised for the PT and CBR-RC primitives, the number of PT and CBR-RC training vectors reported are determined using algorithms implemented in software, as previously mentioned, using a "brute force" binary enumeration method. Also given earlier, Table 2.1 in Section 2.2.1 summarizes the number and type of training vectors for the three neural networks studied.

Before the results of training the three neural networks is given in Section 2.3. the following section discusses some of the speed-up methods employed in the training of the neural networks, and Section 2.2.3 presents the training error and training times.

2.2.2 Neural Network Training Speed-up

While it is not the intent here to give a full account of the different training methods used, it may be interesting to some readers to know the effect of some of the training speed-up methods employed. A description of these methods can be found

in [Hay94], which contains further references.

Two speed-up methods are incorporated into the software suite: training vector reordering and asymmetric activation function. With training vector reordering, as the name implies, the training vectors are uniformly reordered after the entire training set is presented to the neural network. Hence, if the training set is presented to the neural network one hundred times, then the training vectors are reordered one hundred times. This speed-up method has a considerable effect on the training problem at hand, decreasing the training time by a factor of about one half.

The speed-up method of using an asymmetric activation function requires the use of a function such as $out = 1.716 \cdot \tanh\left(\frac{2}{3}net\right)$ instead of the logistic (non-symmetric) activation function of Equation (A.1) in Section A.1.2 of Appendix A. When the asymmetric activation function is employed the zeros of the training vectors are changed to negative ones, that is "1 0 1 0 1 0 1 0 1 0" becomes "1 -1 1 -1 1 -1 1 -1 1 -1." Also, the synaptic weights and threshold values of neurons are initialized with respect to the fan-in of a given neuron, uniformly distributed over the range $\left(-\frac{2.4}{\mathrm{FI}_m^l}, +\frac{2.4}{\mathrm{FI}_m^l}\right)$, where FI_m^l represents the fan-in of neuron m in hidden layer l, following the notation in Section A.1.3. In this context, the fan-in of a given neuron is equal to the number of synapses which terminate on the neuron. Unfortunately, this speed-up method does not reduce the training time by an appreciable amount, nor does it reduce the complexity of the neural network used to learn the given problem. The next section discusses the amount of computer time required to train the three neural networks presented, and shows the training error.

2.2.3 Training Error and Training Times

This section briefly discusses the training error and training times of the 10–35–35–9, 15–80–80–10 and 20–200–200–11 neural networks. As stated, the neural networks are trained using the Backpropagation algorithm derived in Section A.2 of Appendix A. The training methodology used, including a description of the measure Mean Squared Error, is summarized in Appendix B. From the previous section, both speed-up methods of training vector reordering and asymmetric activation function are employed. The number of training vectors for each of the three neural networks is summarized in Table 2.1 of Section 2.2.1.

The Mean Squared Error of the Backpropagation training algorithm is plotted versus the training epoch in Figure 2.14 for the three neural networks in question. As can be seen, the training of the three networks converges in about 3,000 presentations of their respective training sets. This level of training represents approximately zero classification errors, as summarized in Table B.1 in Section B.1 of Appendix B. The reader may note the sudden drop in training error of the 20–200-200-11 neural network after training epoch 2,000. This corresponds to a change in the training parameters of acceleration and momentum (see Appendix A). In effect, the training discovers a (local) minimum in the training vector error surface function.

As far as training times are concerned, the 10–35–35–9 and 15–80–80–10 neural networks require a reasonable amount of time to train—approximately two hours and two days, respectively. Unfortunately, the 20–200–200–11 neural network requires in excess of a month to train, which makes investigation and experimentation

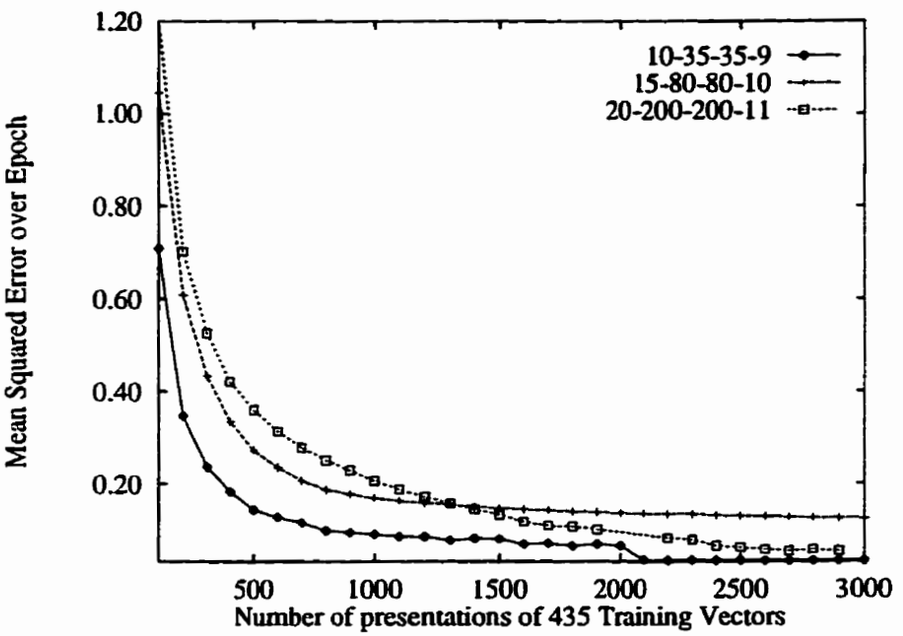


Figure 2.14: Training Error for the 10–35–35–9, 15–80–80–10 and 20–200–200–11 Neural Networks

difficult at best. For further studies to be continued along these lines, this problem will require attention. A more complete discussion of the training times, including the machines used, appears in Section B.3 of Appendix B.

With the training vectors specified and the neural networks trained, all that remains is to validate their training. This is performed in the next section. Then, Section 2.4 describes the operation of the primitive classifier on traffic streams not encountered during training.

2.3 Primitive Classifier Training Results

In this section the training of the primitive classifier based on the 10–35–35–9, 15–80–80–9 and 20–200–200–11 neural networks is validated. Since validating the entire training set would require a multitude of figures, only a few examples of each traffic class is given. Section 2.3.1 shows that the primitive classifier is able to detect CBR streams. Since DG primitives are defined to occur when sources transmit at low rates, this is also shown in this section. Next, a few CBR-RC primitive classifications are plotted in Section 2.3.2. Finally, Section 2.3.3 shows that the primitive classifier can detect streams from the PT traffic class.

As discussed in Appendix B, since all possible traffic primitives from the discrete partition described in Section 2.1.2 are used to define training vectors in the training set, it is not possible to define a validation set as is usual in the literature. Thus, to validate the training, the neural networks will be presented with deterministic traffic streams of the appropriate type.

Before the results are presented, a note should be made about the output of the classifier. As stated in Section 2.2.1 and presented in Appendix C, the output is binary encoded, which is not very human-readable. Hence, considering the 10–35–35–9 neural network based primitive classifier, for each primitive class of the training vectors listed in Tables C.1–C.4 in Appendix C, a number is assigned, which is shown in Table C.5. While the training vectors are not listed, the classifications for the 15–80–80–10 and 20–200–200–11 neural networks are also assigned numbers, which are displayed in Tables C.6 and C.7. It is this number which is on the y-axis of the figures to follow. However, wherever possible, text is inserted.

Due to the fact that the training vectors for the 10–35–35–9 neural network based primitive classifier are made available in Appendix C, this neural network is studied in more detail than classifiers based on the two larger neural networks trained. Nonetheless, in most cases comparisons are made as to the effect of the traffic window size W on the classifications.

2.3.1 CBR Traffic Sources

In this section the results of presenting the three classifiers with CBR sources of various transmission rates are reported. Most of the results are discussed with respect to the classifier with the traffic window of size $W = 10$, in the next section. Then, in Sections 2.3.1.2 and 2.3.1.3, only comparisons are made to the classifier with larger window sizes.

2.3.1.1 Traffic Window Size $W = 10$

As shown in Figures 2.15–2.19, the neural network successfully learns to detect CBR streams. Thus, in order to make the validation a little more interesting, and to give examples of how traffic classification might react at boundary conditions, it is assumed that the CBR sources are just starting transmission. Hence, initially, the traffic window is filled with slots. Then after thirty slot times, the CBR source starts transmission at its specified rate.

Examining Figure 2.15 for the case of the CBR(0) source, it can be seen that for the first thirty slots the primitive classifier correctly classifies this source as type DG. This behavior is normal, since as defined in Section 2.1.3.4, an idle link

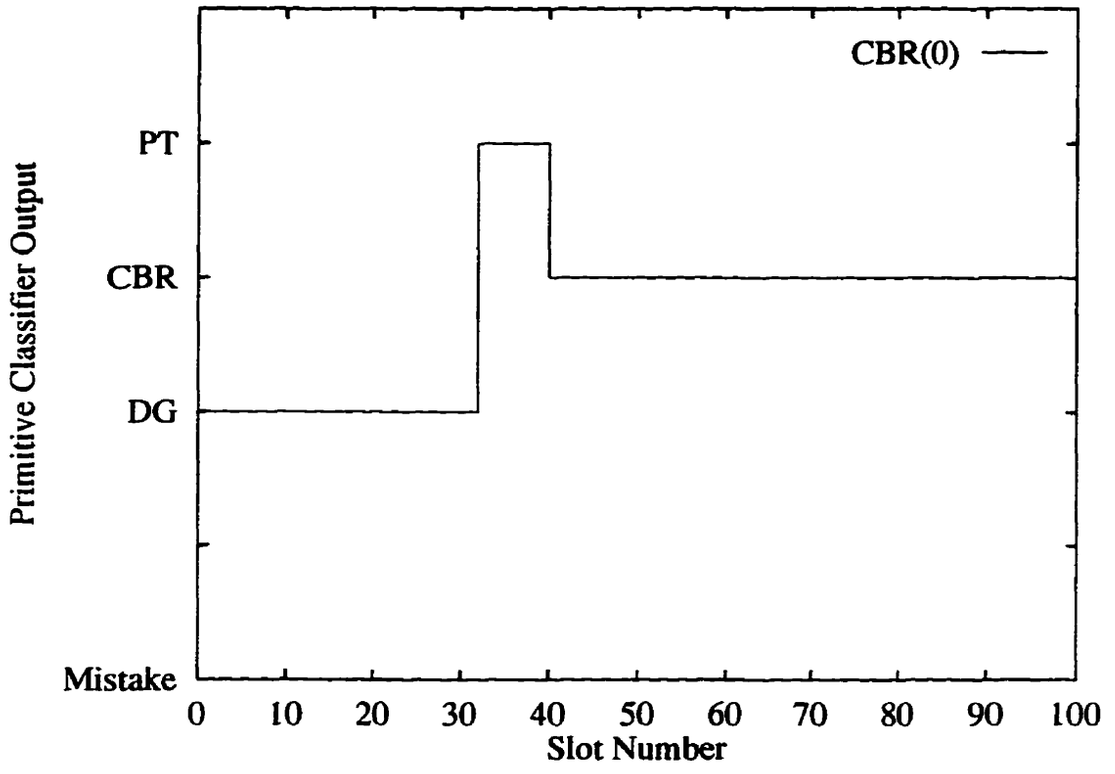


Figure 2.15: Classification of the CBR (0) Source for the 10–35–35–9 Neural Network Based Primitive Classifier

is in the DG primitive class. Then, after two cell arrivals from the source which is now transmitting at the link rate, the classification changes to PT. Again, this is as expected, since the traffic window consists of the "0000000011" slot and cell pattern. This is a morphism of the PT $(2, 8)$ packet train class, as defined in Section 2.1.3.3 and presented in Table C.3 of Appendix C, and so again the classifier is behaving properly. At slot number thirty-nine, the window consists of "0111111111," which is a morphism of the PT $(9, 1)$ packet train primitive class. Finally, after the arrival

of a cell in slot forty, the classification changes to CBR and remains there. Now the window contains all cells, which is defined to a CBR source. Hence, after an initial transient period of ten slots, which happens to be the size of the traffic window for the primitive classifier employing the 10–35–35–9 neural network, the classification is as expected. Note that if the source starts to transmit immediately at the full link rate instead of being initially idle, the transient period would vanish. As stated, this transient period is included so that the validation contains meaningful examples of the classifier operation.

Figure 2.16 shows the classifier operation for the CBR(1) source which transmits at half the link rate. After a short, six slot transient period during which the source makes its transition from the idle state, the primitive classifier correctly detects the source to be CBR. At slot thirty-three, the classifier makes an error. This is the only error the classifier can make, since the neural network training resulted in only one error—and this is it. With the traffic window containing "0000000101," which is defined to be a RC $\frac{C}{8} \rightarrow \frac{C}{7}$ primitive as tabulated in the very last row of Table C.4 of Appendix C, the neural network outputs the binary code "110000000." This does not correspond to any of the defined classes. Since only one error results after training, and considering the desire to keep the neural network small, it is felt that this is acceptable. Corrective action could be taken in additional hardware or software external to the neural network.

The following classifications of Figure 2.16, as the traffic window fills with the cell and slot pattern of the CBR(1) source, show a definite pattern in the transition. This was designed into the classifier by defining and assigning priority to

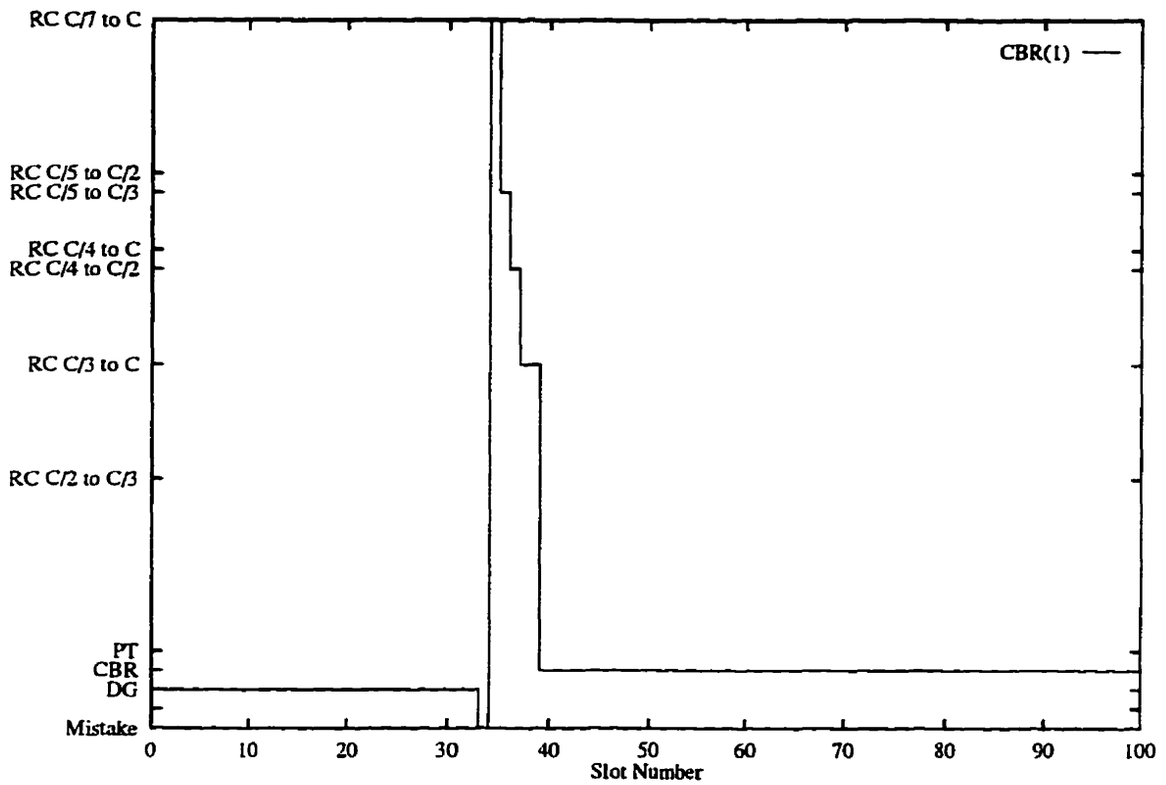


Figure 2.16: Classification of the CBR(1) Source for the 10–35–35–9 Neural Network Based Primitive Classifier

the CBR-RC primitives in a certain manner. For this work, the RC-CBR primitives lead to training vectors which favor transitions from low link utilizations to higher utilizations. In this manner, if a source attempts to renege on its Traffic Contract, these actions which could be harmful to other users are detected first. Other schemes include transitions from higher to lower link utilization, or transitions which deviate from a set CBR sources. As seen from the figure, after the nine slots of transition, the classifier behaves as expected.

The classification of the CBR (2) shown in Figure 2.17 behaves in a very similar fashion to the CBR (1) of Figure 2.16, and thus does not require additional comment, except to state that the transitional period again is nine slots.

The transition of the CBR (4) source shown in Figure 2.18 is much more graceful than the previous ones. This is due to the fact that a CBR source transmitting at one fifth the link rate is relatively sparse when viewed through a window ten slots long; its base primitive is "1000010000." Thus, at slot thirty-five, the traffic

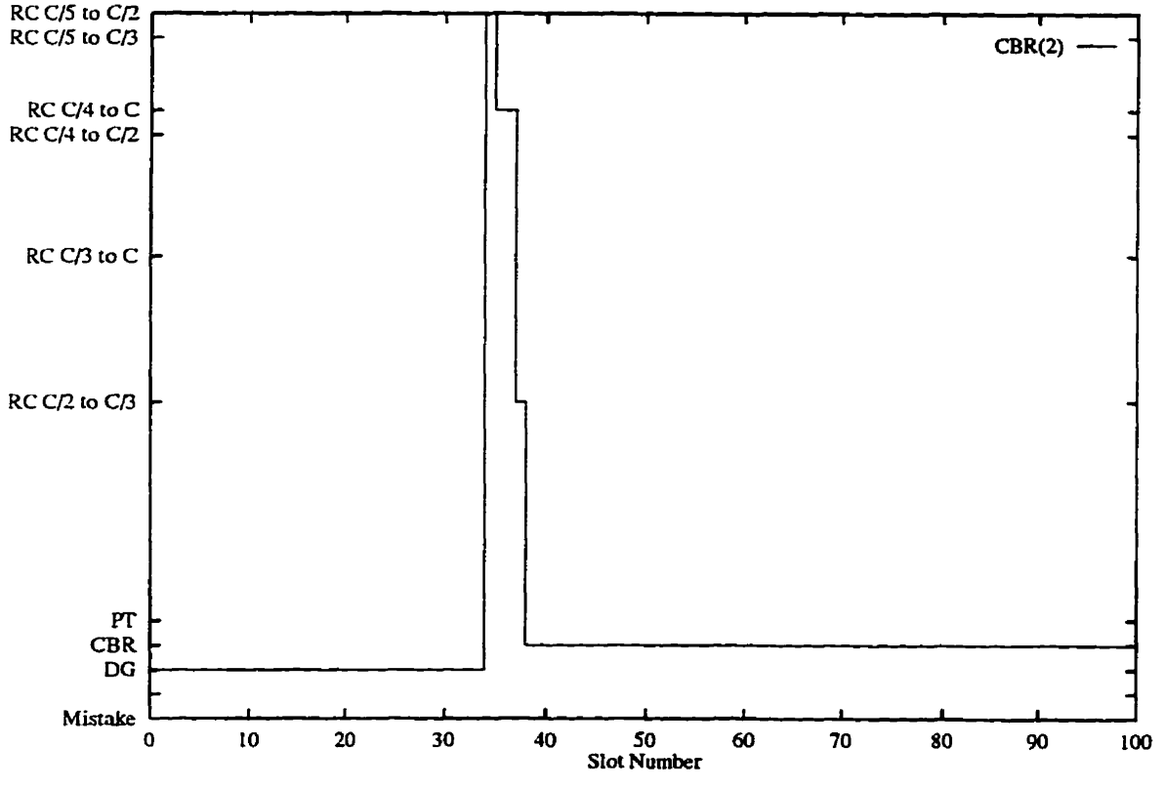


Figure 2.17: Classification of the CBR (2) Source for the 10–35–35–9 Neural Network Based Primitive Classifier

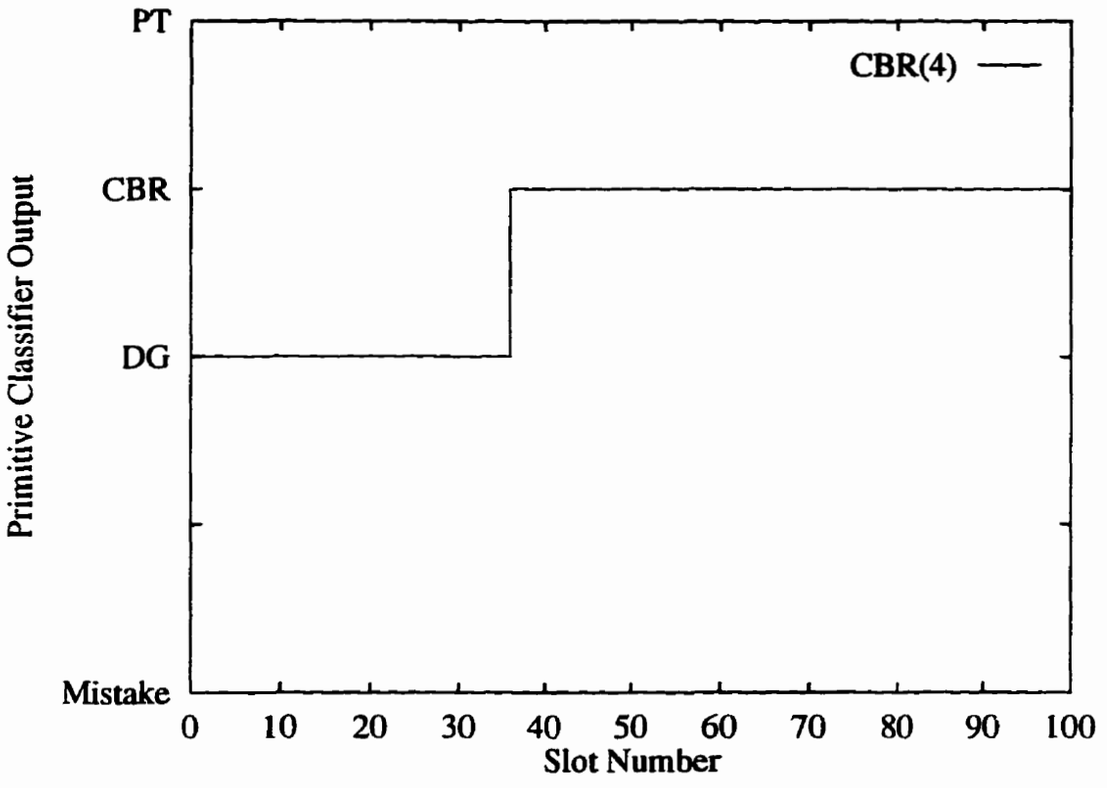


Figure 2.18: Classification of the CBR(4) Source for the 10–35–35–9 Neural Network Based Primitive Classifier

window contains "0000010000," which is in the DG class. Then, starting with slot thirty-six the window contains "0000010001," or one of its morphisms, which is defined to be part of the CBR class.

Figure 2.19 shows the results of classifying the CBR(9) source. As can be seen, and as anticipated in the definition of CBR primitives in Section 2.1.3.1, only CBR sources of rate CBR$(W-1)$ and higher can be classified with a traffic window of size W. Hence, the classification is DG, since the traffic window is filled with either

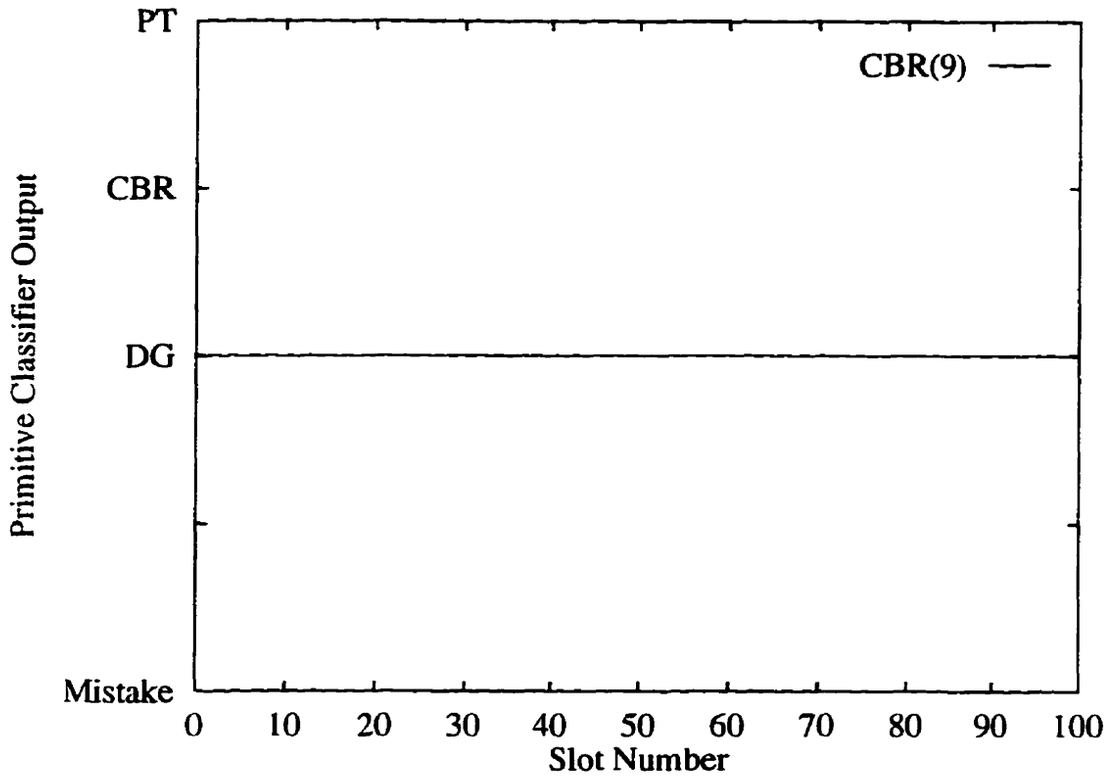


Figure 2.19: Classification of the CBR(9) Source for the 10–35–35–9 Neural Network Based Primitive Classifier

all slots or a cell and all slots. This source has been included for comparison to the primitive classifiers with larger traffic windows, discussed in the following two sections.

The classification results of the above CBR sources, excluding the CBR(14) source, are summarized in Figure 2.20. The way the traffic primitives are designed, sources with higher activity make further "excursions" away from the "known" source classes of CBR, PT and DG, and in general "stay away" for longer periods.

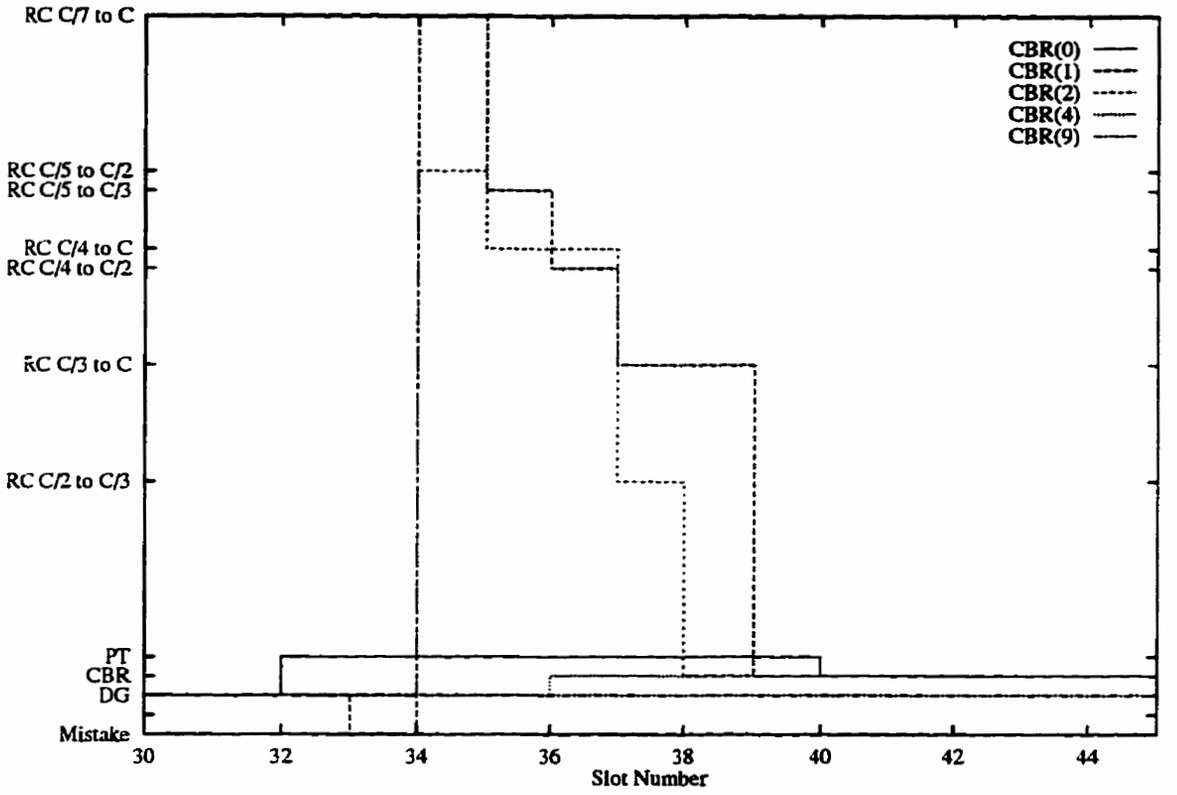


Figure 2.20: Comparison of the CBR Classification Results for the 10–35–35–9 Neural Network

These excursions come in the form of CBR-RC traffic primitives. Thus, the CBR (1) source which utilizes one half of the link rate makes the largest excursion RC $\frac{C}{7} \rightarrow C$, whereas the CBR (4) does not make any excursions at all. These results are now compared with the two larger window sizes, in the following two sections.

2.3.1.2 Traffic Window Size $W = 15$

The validation results for the primitive classifier based on the 15–80–80–10 neural network do not differ substantially from those given in the last section. The only major difference is that since $W = 15$ in this case, the transition period as the CBR sources change their rate from idle to the appropriate level is, in general, five slots longer. For example, Figure 2.21 shows the case of the CBR(1) source. The y-axes of Figure 2.21 and the two following figures contain the number associated

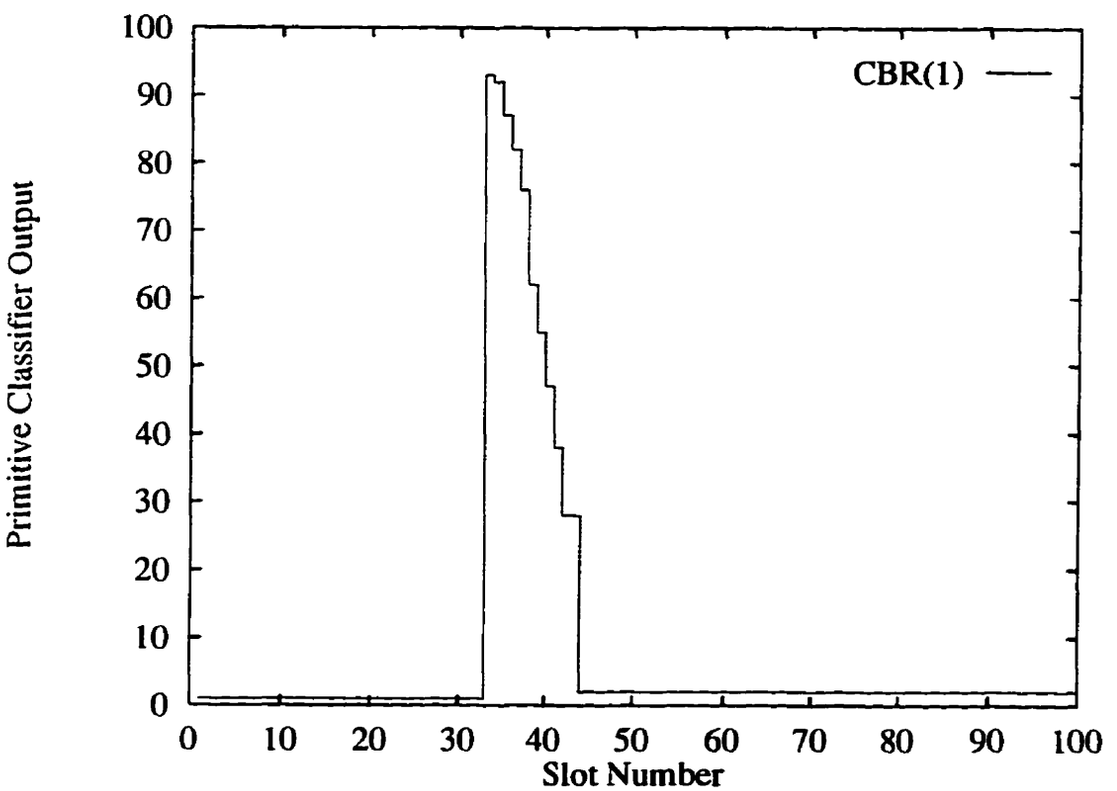


Figure 2.21: Classification of the CBR(1) Source for the 15–80–80–10 Neural Network Based Primitive Classifier

with the classification, the human-readable format of which which can be found in Table C.6 of Appendix C. To the left of the "transition peak" of Figure 2.21, the classification output is one, which corresponds to the DG class. To the right of the peak, the output is two, or CBR, the desired classification. At the peak of the transition period, the output is ninety-three, that is a RC $\frac{C}{13} \rightarrow \frac{C}{12}$ classification. Note that the length of the transition period is eleven slots, five more than in the $W = 10$ case, which corresponds to the increase in size of the traffic window. In addition, while the excursion to the highest point in the peak comes as a single step, as in the case with $W = 10$ of Figure 2.16, the "fall" from the peak is more gentle, and goes through more classifications. In other words, this larger window size is able to make more, or finer classifications than the smaller window. Thus, the results support the design discussion of Section 2.1.1. Finally, note that since the 15–80–80–10 neural network completed training with zero errors on the training set, the mistake that was observed in Figure 2.16 is not present in Figure 2.21.

The classification results for the same sources that are presented in the previous section appear for the present case in Figure 2.22. The same transition patterns are observed, except as noted they have increased in length and accuracy, or "characterizability." Also note that the CBR(9) source which was classified as DG in Figure 2.19 for $W = 10$ of the last section is now classified correctly as CBR. As a point of interest, for $W = 15$, it is now the source with the most graceful transition.

2.3.1.3 Traffic Window Size $W = 20$

Finally, a summary of the validation results of the CBR sources for the 20–200–200-11 neural network based traffic classifier is shown in Figure 2.23. As can be seen,

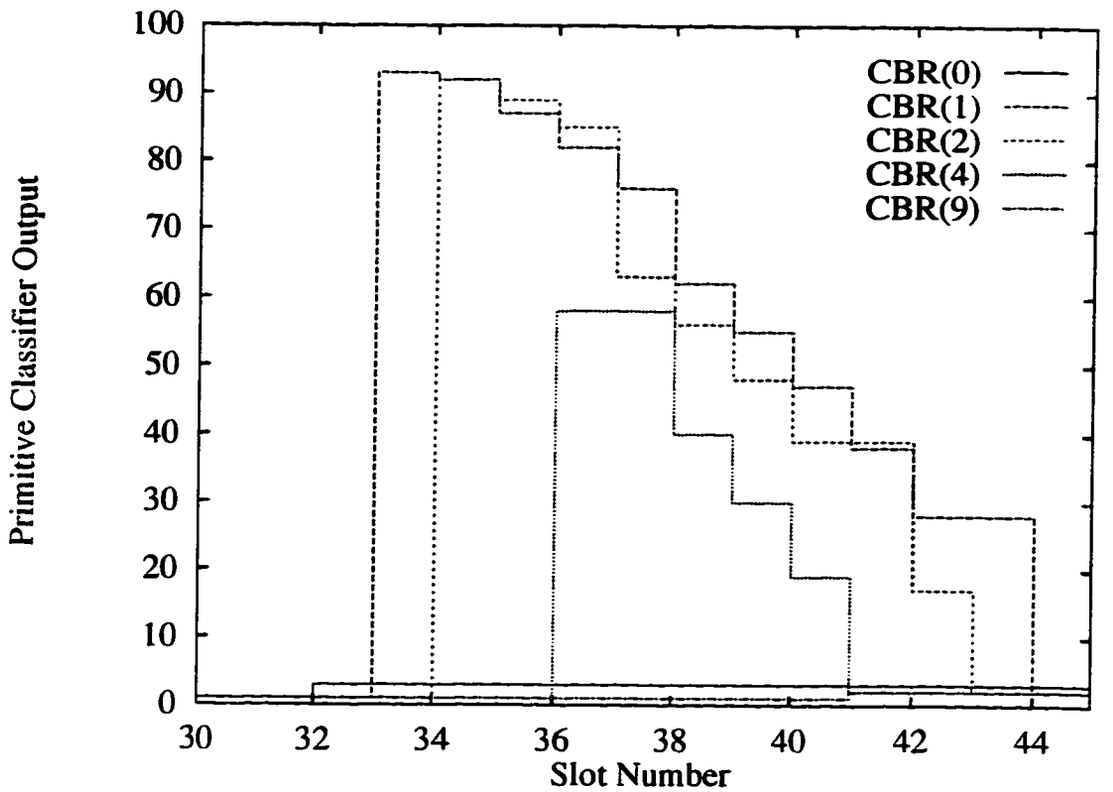


Figure 2.22: Comparison of the CBR Classification Results for the 15–80–80–10 Neural Network

the transition period has spread out once again, by five slots over the $W = 15$ case and ten slots over the $W = 10$ case. In addition, the classifications have become even finer, with over 160 possible. In this case, the CBR (9) has retained its graceful transition characteristics, and is now joined by the CBR (14), which is not plotted. However, they have different lengths of transition period. Finally, note that since this neural network trained with nine mistakes, one of these has been encountered by the CBR (4) source.

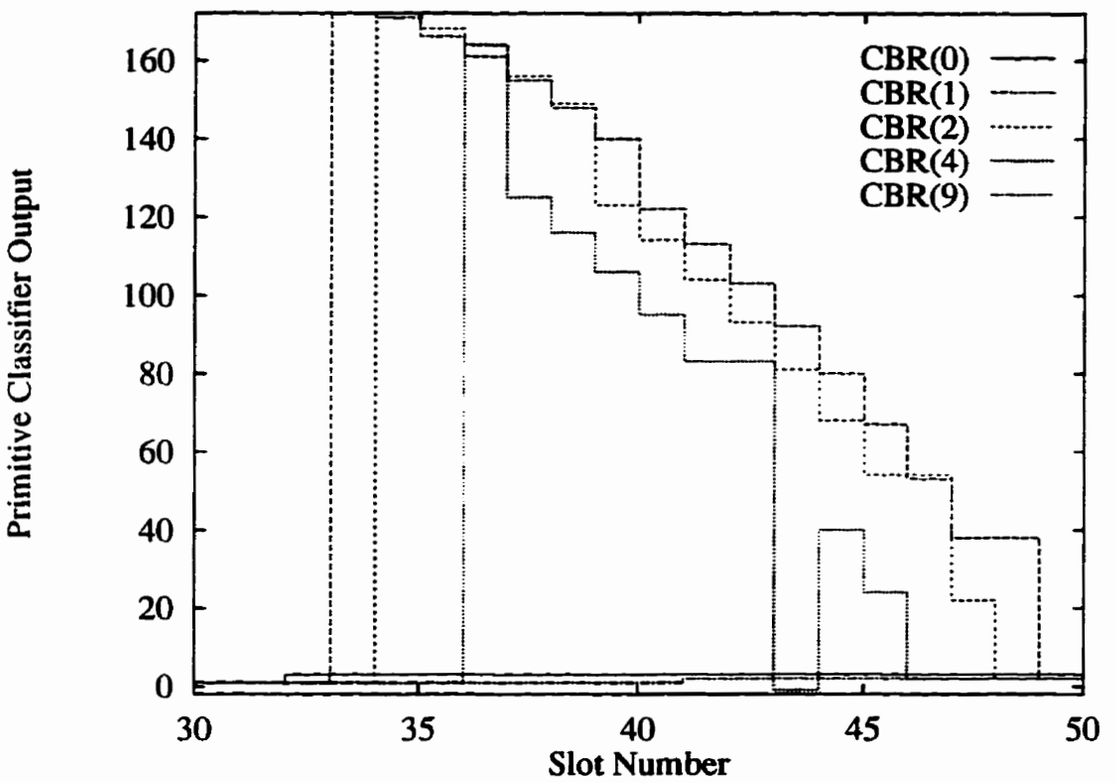


Figure 2.23: Comparison of the CBR Classification Results for the 20–200–200–11 Neural Network

While this section is intended to discuss the neural network training validation results of CBR sources, due to the design of the traffic primitives, this section also included all that needs to be stated about the DG class and much about the CBR-RC class. The following section makes a few more points about the training results of the CBR-RC sources, and Section 2.3.3 briefly presents the results of the PT sources.

2.3.2 CBR-RC Traffic Sources

In addition to the discussion of the CBR source "transitions" of the previous section, a few points regarding the training validation results are made here. The reader should be aware that the CBR source transitions are nothing more than a sequence of CBR-RC classifications, as the figures of the previous section show. This is not surprising, naturally, since by definition the CBR-RC primitives are designed to detect the occurrence of a CBR source changing its rate. In the previous section, the transition is from an idle link to the transmission rate of the CBR source. As shown, many CBR-RC classifications are generated as the sources make their transition to their transmission rate. The last section also shows the effect of the traffic classifier window size: the classification accuracy increases as the window size increases, as seen by the gentler fall from the peak excursion through more classifications, but this comes at the cost of increased reaction time of classification, since the transitions occur over a larger number of slots. As discussed in Section 2.1.1, this is the expected behavior of the traffic classifier.

In order to validate specific CBR-RC transitions, a single snap-shot of a traffic stream is presented to the classifier. For example, if "0101011111" is presented to the $W = 10$ classifier, the classification is RC $\frac{C}{2} \rightarrow C$, as expected. If this is continued with the remainder of the CBR-RC primitives, only one mistake will be made, as discussed in the previous section. The $W = 15$ classifier does not make any mistakes, and the $W = 20$ makes nine. Since the CBR-RC classifications are designed to detect changes in CBR sources, further results appear in Section 2.4.

2.3.3 PT Traffic Sources

Since all the PT training vectors are learned perfectly for all three sizes of neural network, validation results are difficult to present; the figures all consist of straight lines indicating the output of the classifier is PT for any traffic stream that has the characteristics of the PT (x, y) source defined in Section 1.6.6.4. Of course, this is true only for PT sources which are characterized by the PT primitives of Section 2.1.3.3; that is, any PT source which is completely contained within the traffic window. This idea is discussed next.

Two interesting figures are presented which illustrate a concept that is useful in Section 4.2 of Chapter 4. Consider Figure 2.24, which represents the output of the $W = 10$ classifier when presented with thirty slots, and then a PT $(10, 10)$ source. After the first thirty slots, the first classification made is DG, since as the burst of this PT source enters the traffic window, a DG traffic primitive results, namely "0000000001." Then, for the next nine slots, the classification is PT, since the traffic window is filled with morphisms of various PT traffic primitives. Note, however, due to the definition of the PT primitives, there is no PT $(10, 10)$ traffic primitive for $W = 10$. Thus, at slot thirty-nine, the traffic window is "0111111111." At slot forty, a CBR classification is made, since the traffic stream now resembles the CBR (0) source. At slot forty-one, the output is once again PT and this cycle repeats every twenty slots. In a similar fashion, at slot fifty, the traffic window contains only slots, and so the classifier output is DG, and this observation is also repeated every twenty slots. Hence, as anticipated in Section 2.1.1, the ability of the primitive classifier to discern features in a traffic stream is directly related to

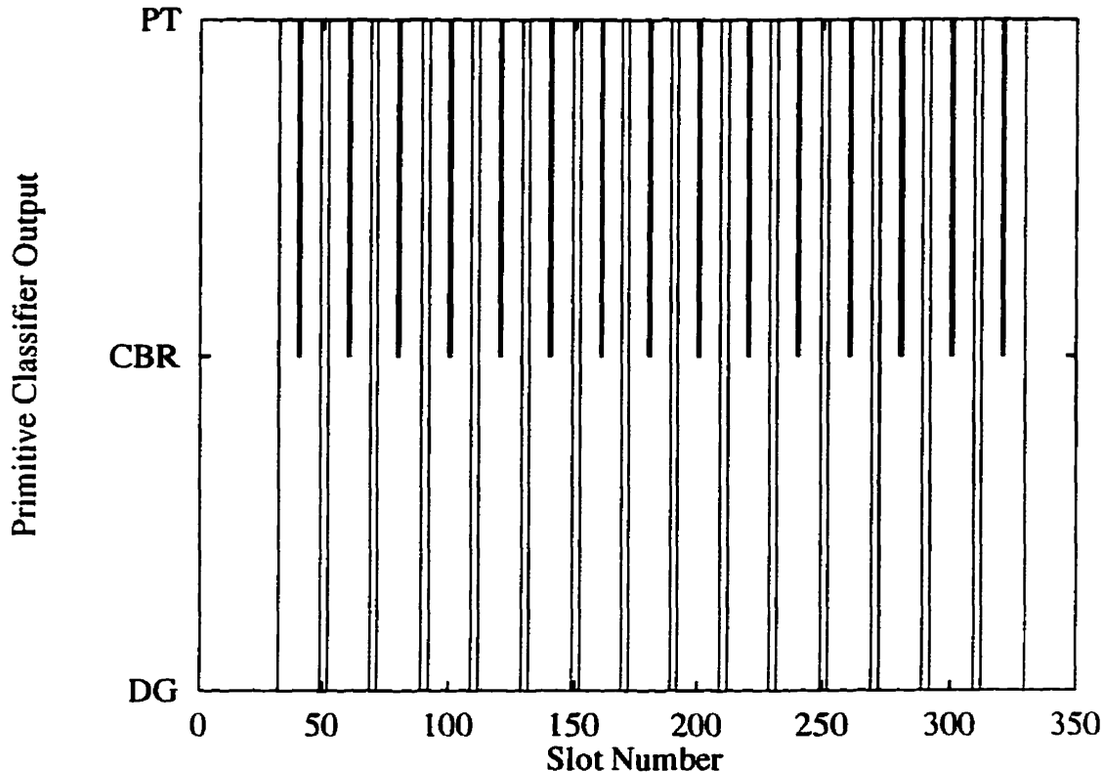


Figure 2.24: Classification of the PT (10, 10) Source for the 10–35–35–9 Neural Network Based Primitive Classifier

its traffic window size. As well, the continuity of classifications over time, that is as a new cell or slot enters the traffic window, is also related to the definition of the traffic primitives. In this case, if the traffic primitive "1111111111" is defined to be PT instead of CBR, and "0000000000" is defined to be PT instead of DG, then these results would not be observed. As mentioned before, the results of traffic classification are very much dependent on the desires of the designer of the traffic primitives, that is, the network provider.

This situation is further illustrated in Figure 2.25, which shows the results of the PT(30, 30) source which has much larger bursts and silence periods than the previous source. As can be seen, this only exacerbates the problem, since now the traffic window is completely filled with all cells or all slots for longer periods of time. This problem could be solved by presenting the same stream to traffic classifiers with larger windows, however this represents a large investment in neural network training time, as discussed in Section 2.2.

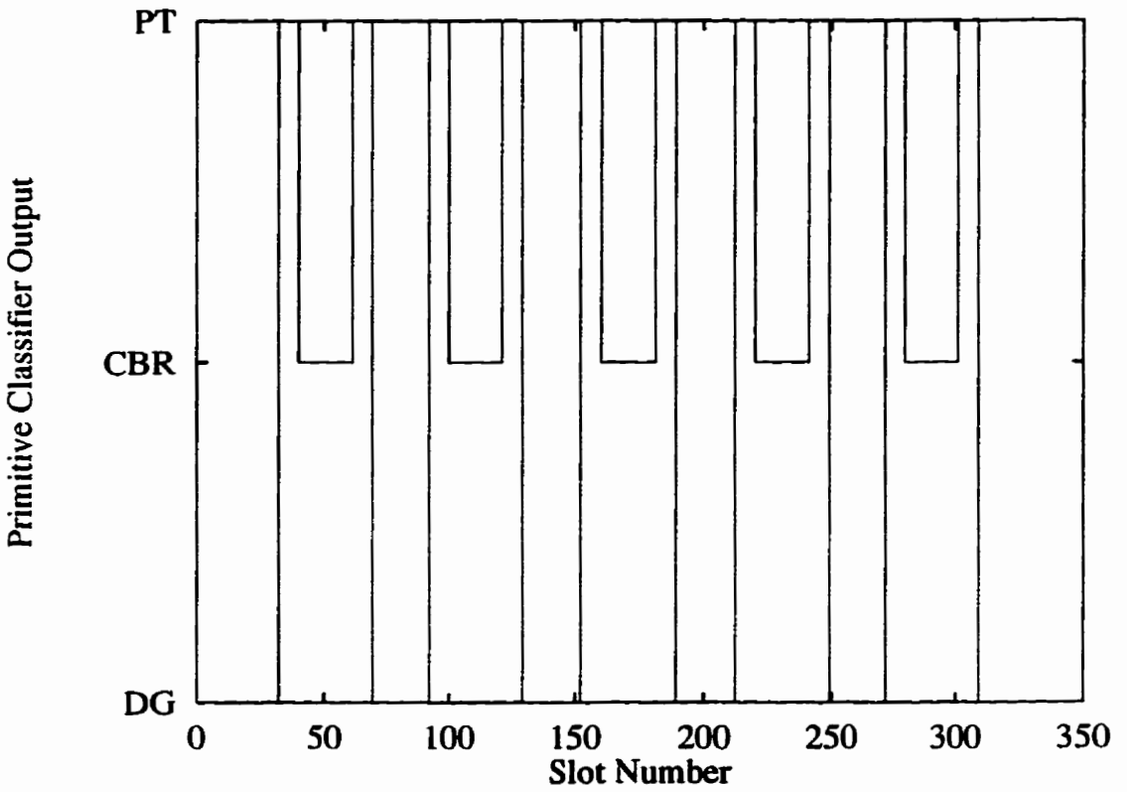


Figure 2.25: Classification of the PT(30, 30) Source for the 10–35–35–9 Neural Network Based Primitive Classifier

This completes the discussion concerning the validation of the training of the traffic primitives. As can be seen, for all three sizes of neural network, the training is successful, and the traffic classifier based upon these neural networks operates as intended. The following section shows the results of the contribution of traffic classification, that is classifying sources which are not presented to the neural networks during training. The above discussion also foreshadows the introduction of the Burst-oriented shaper in Chapter 4, since its design is based on this concept of bursts and silences being contained by the traffic window.

2.4 Primitive Classifier Operation

This section applies the results of training the traffic classifier to traffic streams which its neural network had not been exposed to during training. Since for a given traffic window size W all the deterministic partition traffic primitives are translated into training vectors and used to train the neural network, this implies that all the traffic sources discussed below are drawn from the distribution partition. This represents an important contribution of this thesis: using a neural network that has learned only deterministic traffic streams to detect features in probabilistic streams. And hence the justification for employing a neural network instead of a more conventional classification scheme. Since the power of a neural network is its ability to generalize, by training it to classify deterministic streams it should be able to generalize to probabilistic streams.

The following three sections discuss three such sources of probabilistic traffic streams. First, in Section 2.4.1, general on-off traffic sources and their subset, packet train, are considered. These should be classified well, since in the design of the primitive classifier priority is given to the training of PT primitives in anticipation of the interaction of the classifier and the AAL. Next, Section 2.4.2 briefly examines the performance of the classifier in the face of traffic which, in a way, is the exact opposite of that it learned to recognize. Since the geometric distribution is memoryless, it should pose a serious problem for the primitive classifier. Fortunately, this traffic class is not anticipated after cellization as described in Section 1.6.4, however if the primitive classifier is to have a wider application, this traffic type requires consideration. Finally, Section 2.4.3 presents the results of attempt-

ing to classify a source with characteristics similar to a geometric source, but with the memoryless property no longer holding; in particular, an MMBP source.

2.4.1 On-off and PT Traffic Sources

As a first example, consider the on-off source which has its on-period uniformly distributed with mean five cells, and its off-period uniformly distributed with mean five cells. Using the notation of Section 1.6.6.4, define the uniformly distributed packet train source as $\mathrm{PT}_U(x,y) = \Psi_{OO}(\mathcal{U}(x), \emptyset, \mathcal{U}(y))$, so that the on-off source just described can be denoted by $\mathrm{PT}_U(5,5)$. The result of classifying this source appears in Figure 2.26. The first two hundred slots are shown for this source, and the classifier output. As can be seen, most of the classifications are PT, as is desired. Hence, the neural network has generalized as planned. The CBR and DG classifications arise due to the fact that some of the burst lengths are longer than the traffic window, and that some of the silence lengths similarly are also longer, as discussed in Section 2.3.3. In addition, there are a number of CBR-RC primitives, which are undesirable. They arise due to the characteristics of the $\mathrm{PT}_U(5,5)$ source; it is possible for one cell to appear in a "burst," and one slot in a "silence." However, a pattern such as this is defined to be CBR, and thus as it traverses the traffic window it is inevitable that a CBR-RC pattern is formed. Another source of the CBR-RC classifications could be "boundary conditions." What is meant by this is that as an old burst shifts out of the traffic window as a new burst shifts in, the pattern may in fact resemble a CBR-RC, especially with one or two single cells near the center of the window. Thus, owing to the nature of

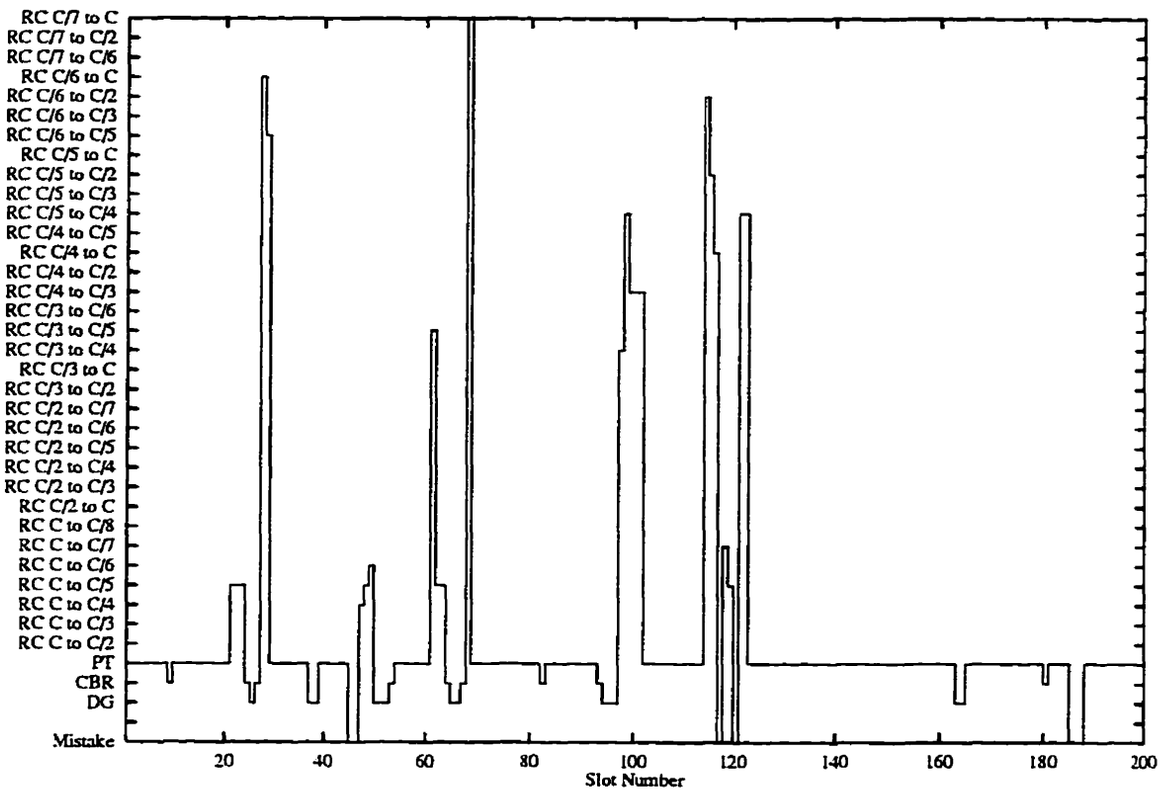


Figure 2.26: Source: $PT_U(5,5)$. Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

the traffic primitives, a certain amount of incorrect classification can be expected and thus should be tolerated. Also note that since the 10–35–35–9 neural network is employed, mistakes can occur. In order to better quantify the classifications, Figure 2.27 plots a histogram of the classifications made. The figure shows that about 86% of the classifications are PT, which certainly does not seem to be the case from the misleading Figure 2.26. In addition, less than 0.1% of the classifications are CBR and DG combined. However, 2.7% of the classifications are mistakes.

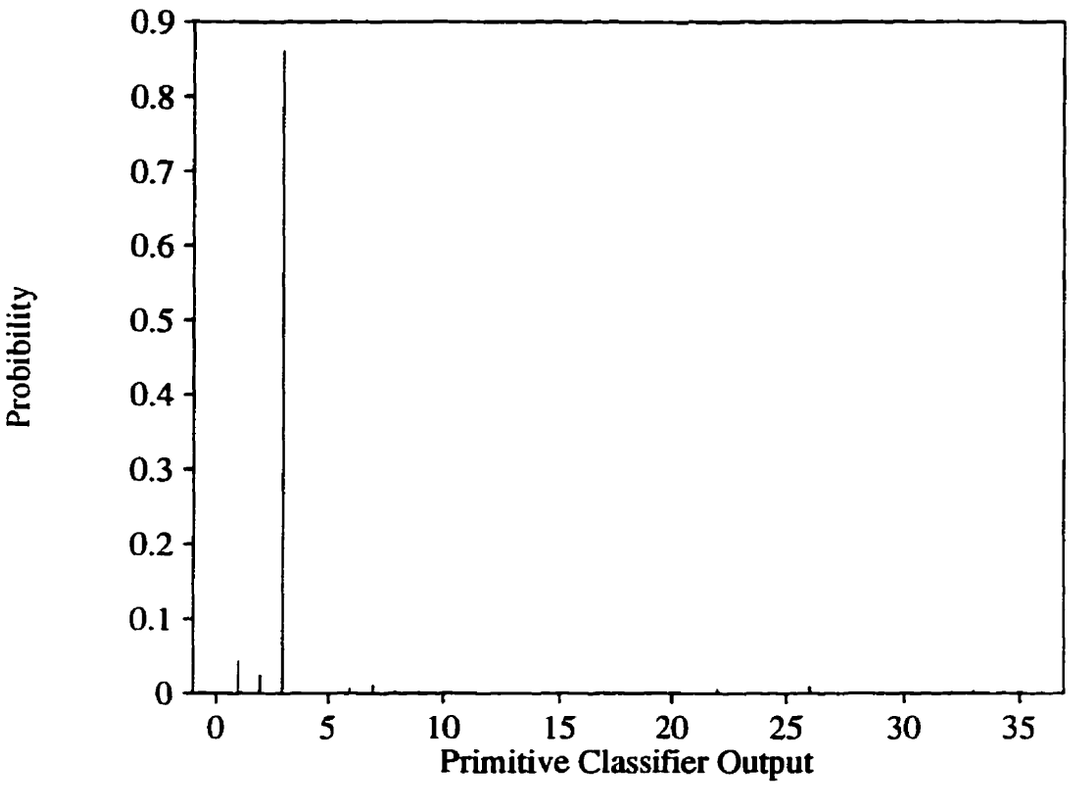


Figure 2.27: Source: $PT_U(5,5)$. Histogram of Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

Thus, perhaps this neural network should "go back to school" in order to learn the single training vector which causes these mistakes. The remaining ten percent is spread amongst the CBR-RC primitives, with no single one representing more than 0.01% of the classifications. Therefore, notwithstanding the training error, it can be said that the primitive classifier operates exceptionally well, correctly discerning a PT cell stream from a somewhat random on-off stream.

A more random on-off source is given by one which has geometrically distributed

on and off periods. Define such source as $\mathrm{PT}_G(x,y) = \Psi_{OO}(\mathcal{G}(x), \emptyset, \mathcal{G}(y))$. The classification of a $\mathrm{PT}_G(5,5)$ source is shown in Figure 2.28, and its corresponding classification histogram in Figure 2.29. As can be seen from the classification output, there are some very interesting features, but as the histogram shows, the excursions into CBR-RC classifications do not have a high probability mass. In this case, only 63% of the classifications are PT, whereas CBR and DG classifications occur 11% and 16% of the time, respectively. The mistakes occurred 5.7%

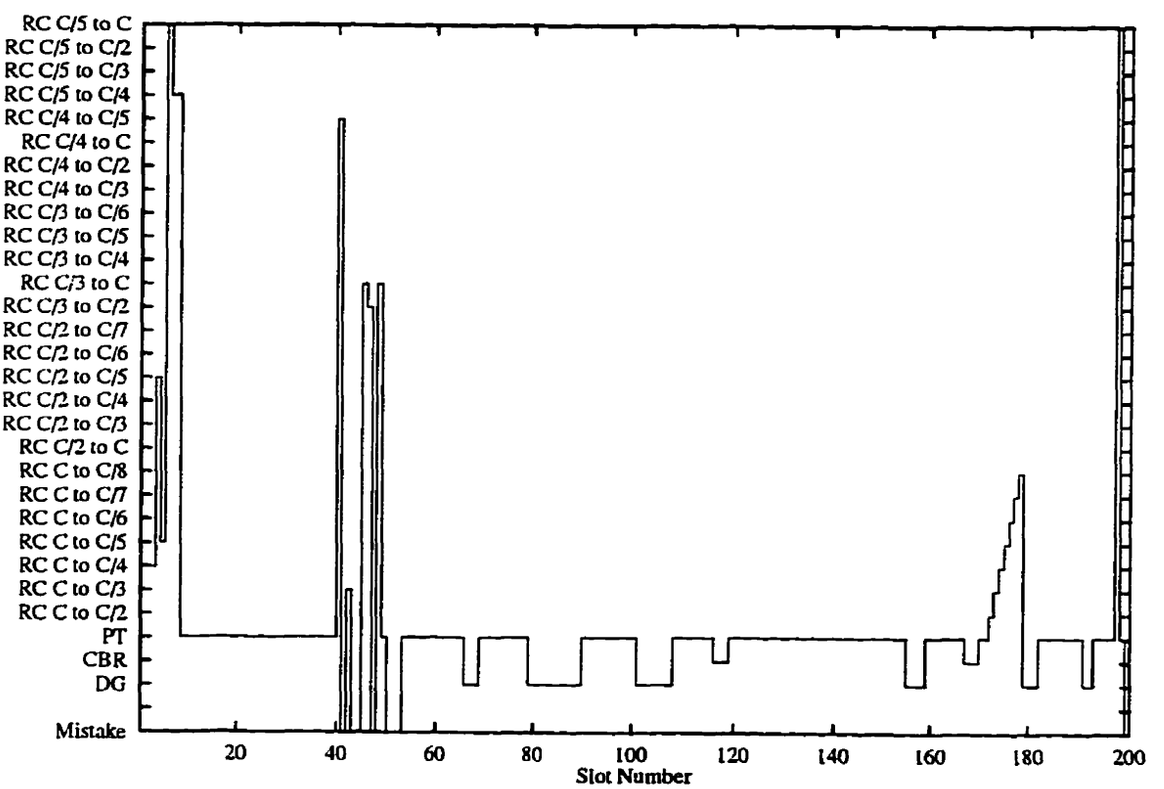


Figure 2.28: Source: $\mathrm{PT}_G(5,5)$. Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

of the time. In addition, the excursion into the CBR-RC primitives is not as far as in the case of Figure 2.26. Thus, while the classification of this source as PT is not as accurate as with the $\mathrm{PT}_U(5,5)$ source, one could make the argument that a $\mathrm{PT}_G(5,5)$ has moved away somewhat from a true packet train source. Hence, perhaps it should not be classified as on-off in the first place. While plots for the larger traffic window size classifiers are not given, the preceding discussions help to develop some intuition. If the burst and silence lengths of a source are small

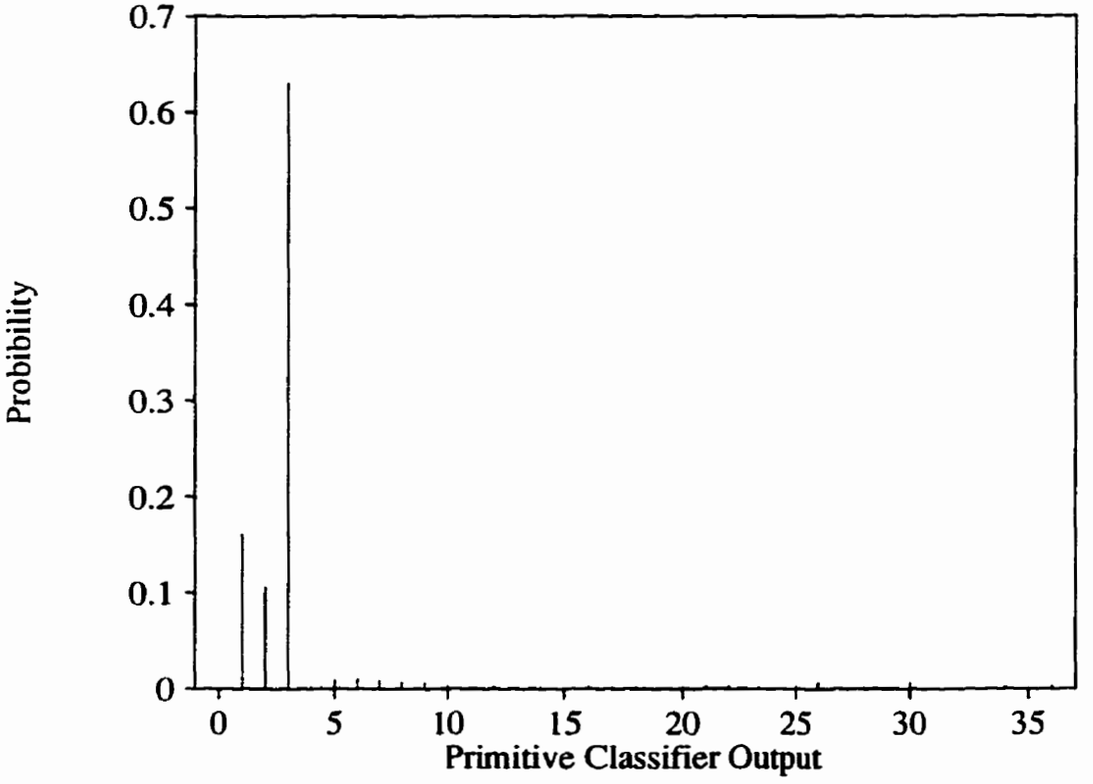


Figure 2.29: Source: $\mathrm{PT}_G(5,5)$. Histogram of Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

relative to the window size, then the larger sized windows do classify the source better, since they observe more of the stream and thus have a higher chance to make a more "accurate" classification. In particular, the DG and CBR classifications are eliminated, but fewer PT classifications are also made. Hence, the larger window size allows more of the stream to be observed, and also allows more diverse traffic characterization. However, if the relative burst and silence lengths are large, then the traffic window does not contain enough characteristics for a good classification to take place. Test results (not included here) indicate that the window size should be on the same order as the sum of the mean burst and mean silence lengths expected.

The $\mathrm{PT}_G(5,5)$ source resembles a PT stream much less than the $\mathrm{PT}_U(5,5)$ source. In the following section, a source type which does not resemble a packet train at all is presented to the primitive classifier.

2.4.2 Geometric Traffic Sources

This section shows the results of classifying two geometric sources, with fairly different mean arrival rates, in order to show that a high arrival rate geometric source appears to be behave as a PT source, whereas a low arrival rate geometric source appears more like a DG source. This could be of appeal to an analyst who must deal with PT sources, but wishes to apply the simplifying assumptions of a memoryless distribution. As defined in Section 1.6.6.4, define two geometric sources, one with mean interarrival time of two cells, $\mathcal{G}(0.5)$, and the other with mean interarrival time of five cells, $\mathcal{G}(0.2)$. In Figures 2.30 and 2.31, the classification output of the

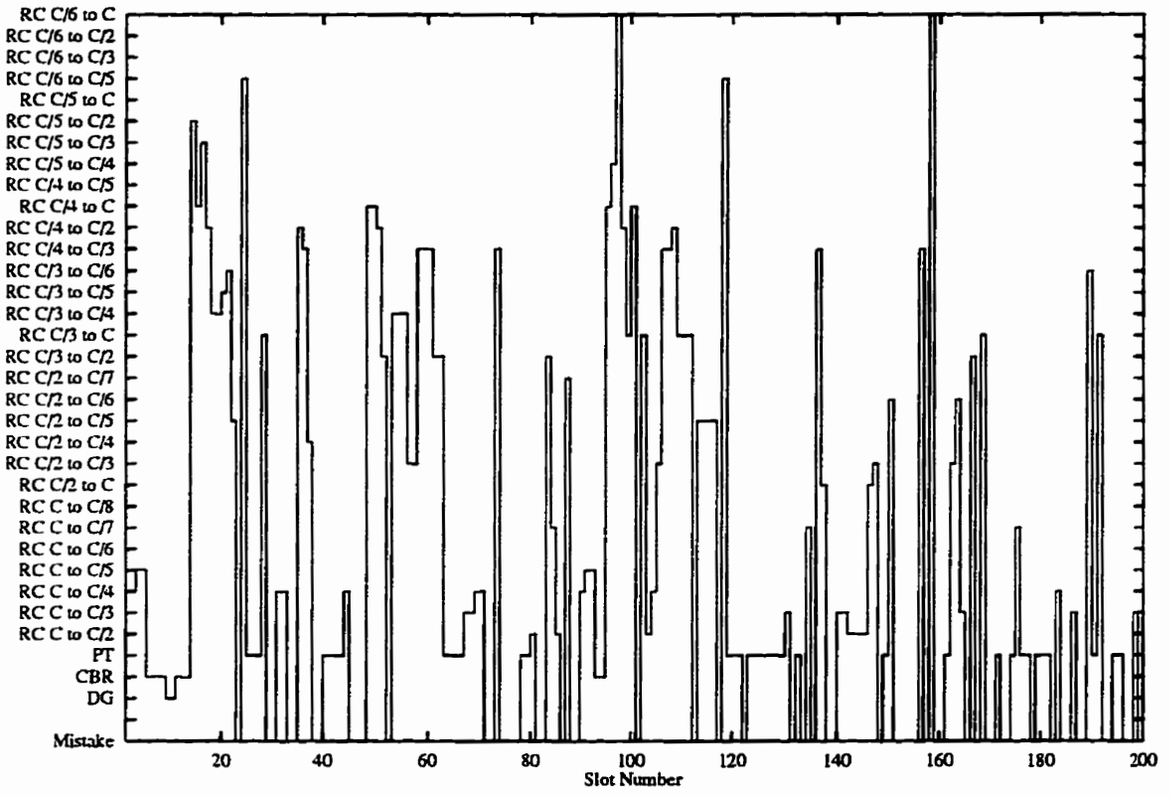


Figure 2.30: Source: $\mathcal{G}(0.5)$. Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

high rate geometric source is shown, along with the classification histogram. As can be seen, a geometric source of this rate appears to resemble a PT source and, in fact, about 30% of the time the classifier does make that judgement. On the other hand, Figures 2.32 and 2.33 show that the low rate source does not resemble a PT source at all, since almost half of the time it appears as a DG source. Of course, since the mean interarrival time is five cells, its "silence" period is starting to approach the size of the traffic window, as discussed previously, the classifier

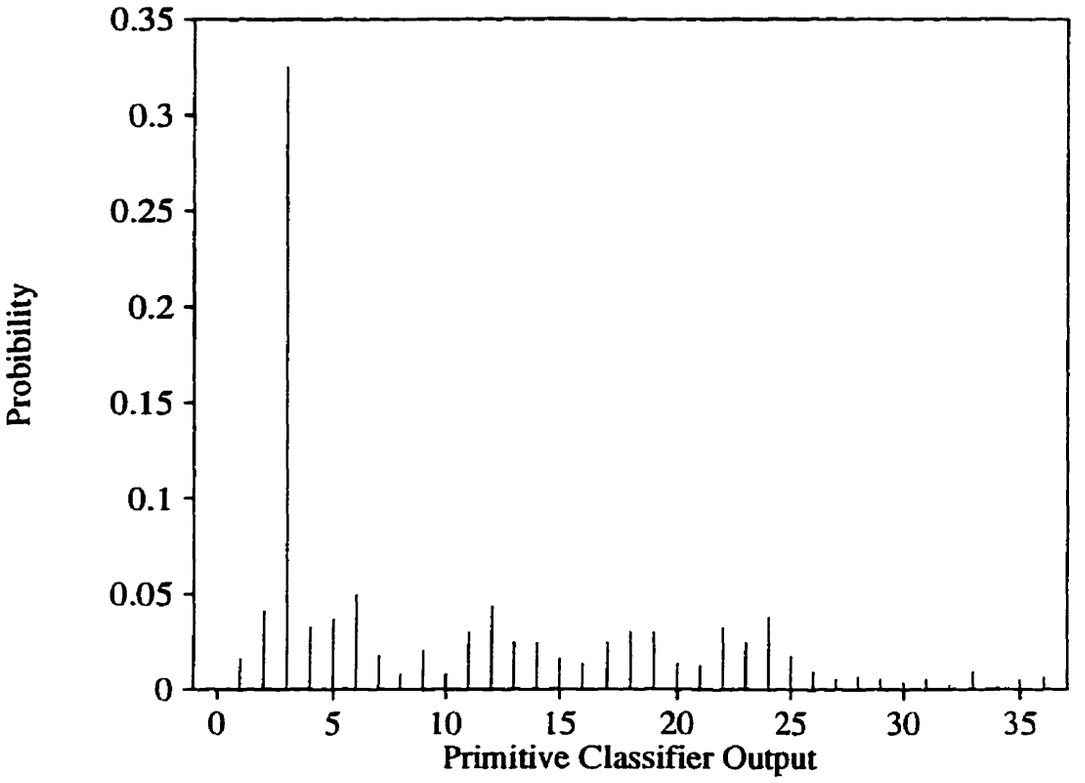


Figure 2.31: Source: $\mathcal{G}(0.5)$. Histogram of Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

may not be able to observe enough of the cell stream to make a good classification.

The result of this section is that while the primitive classifier, as designed, is poorly suited for classifying memoryless sources, such as geometric, it may be useful to distinguish between two types of the same source, as will be discussed in Chapter 3.

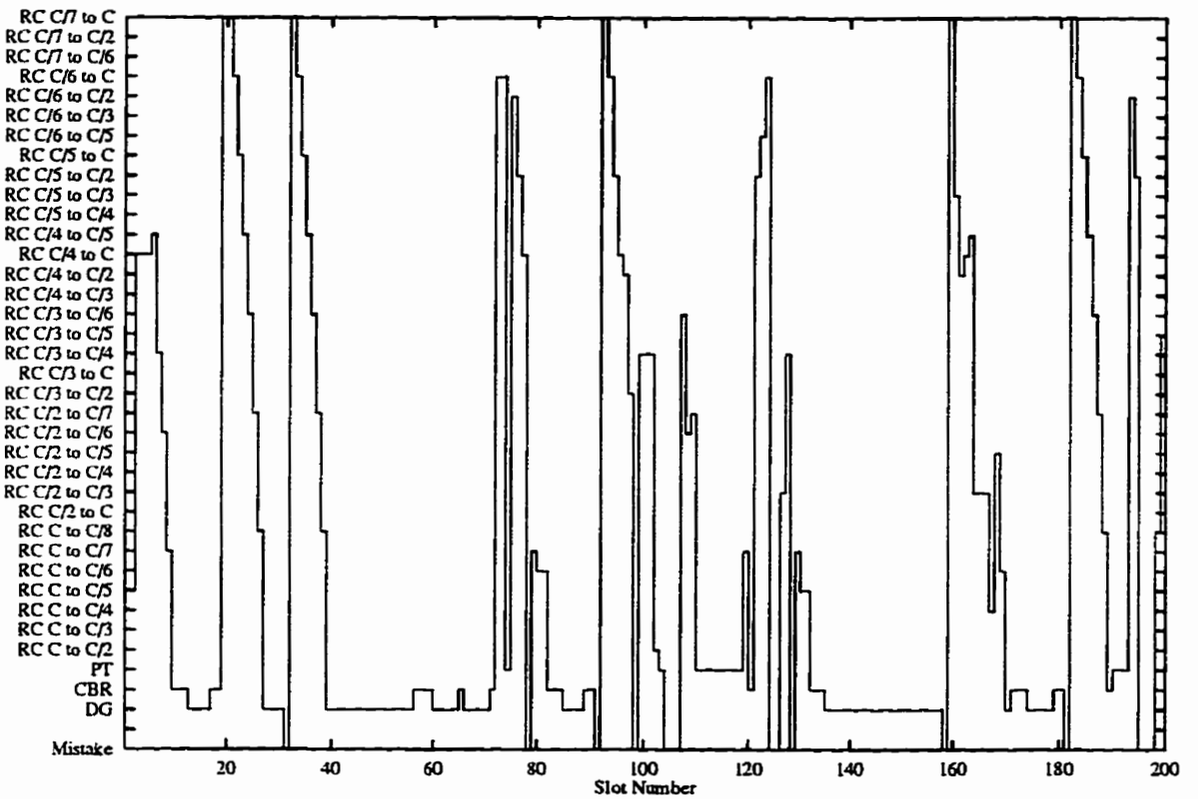


Figure 2.32: Source: $\mathcal{G}(0.2)$. Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

2.4.3 MMBP Traffic Sources

As a last example of traffic classification, Figures 2.34 and 2.35 show the results of classifying an MMBP traffic source, with rate transition matrix

$$\begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix},$$

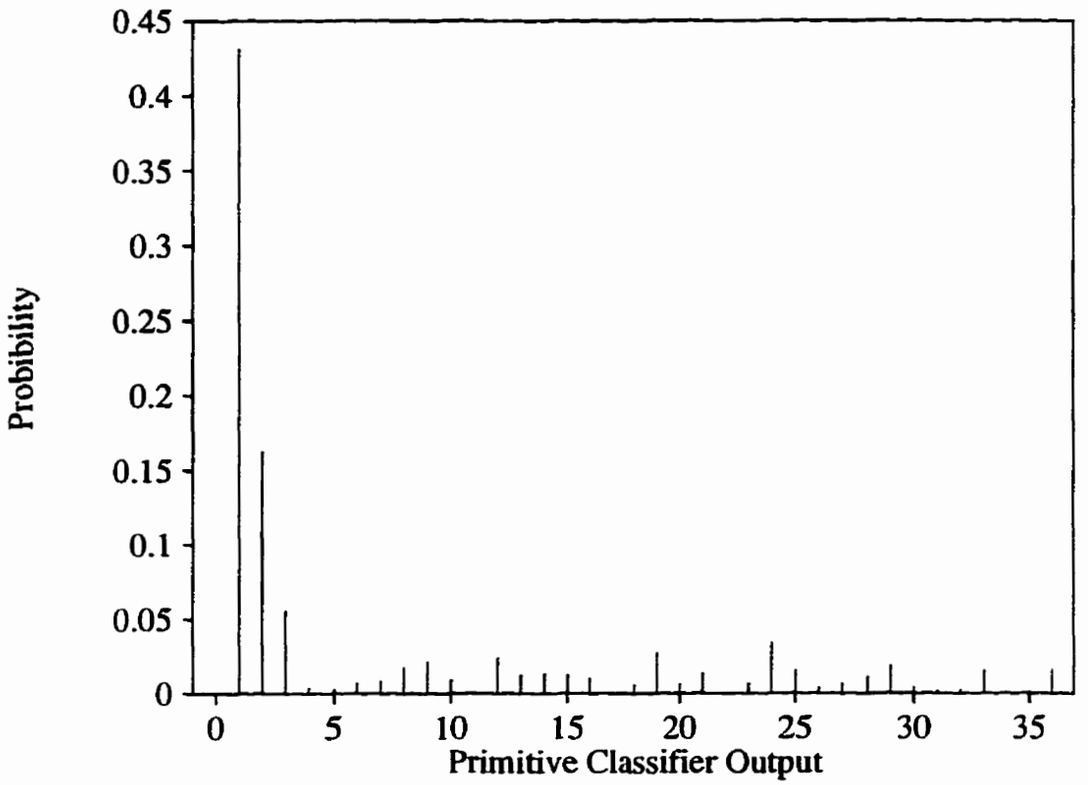


Figure 2.33: Source: $\mathcal{G}(0.2)$. Histogram of Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

and arrival rate vector

$$\begin{bmatrix} 0.9 \\ 0.01 \end{bmatrix}.$$

As expected, the figures show that while this source has some properties of the geometric source, it behaves more like a PT source than the geometric does. About 50% of the classifications are PT, compared with the 30% of the high rate geometric.

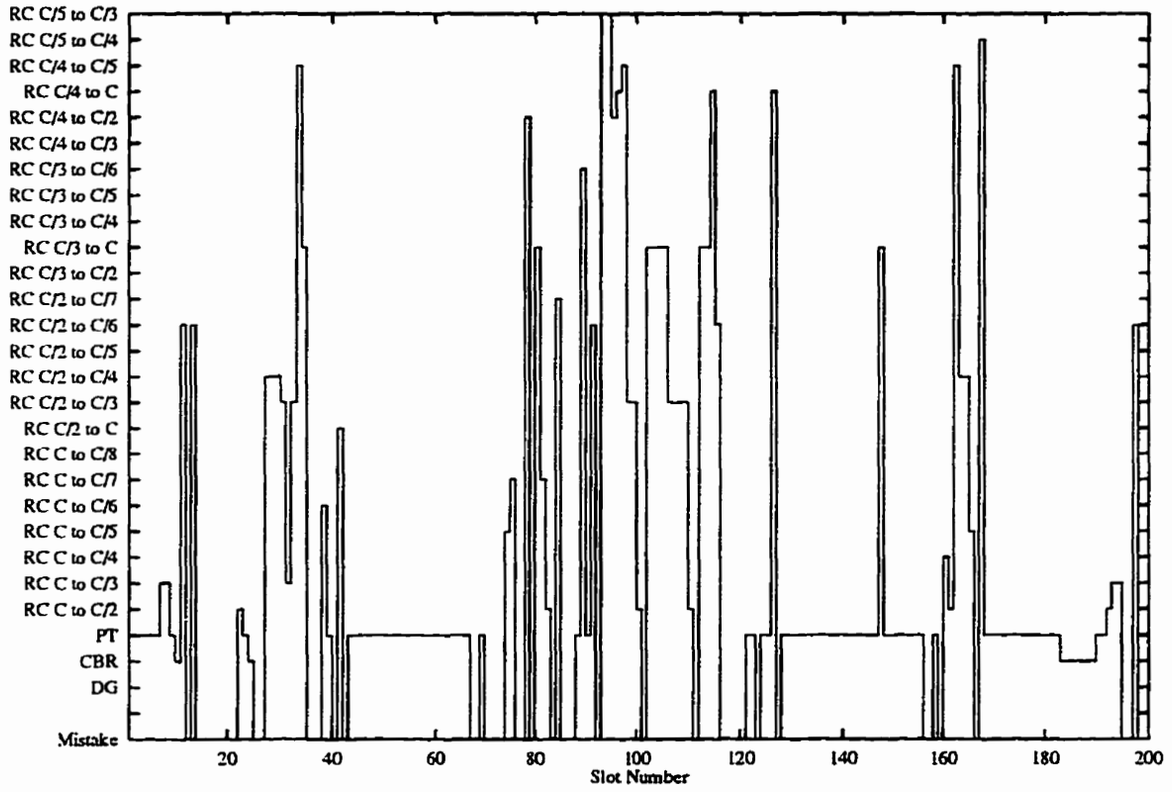


Figure 2.34: Source: MMBP. Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

This concludes the discussion of the operation of the traffic classifier on probabilistic streams. It is shown that for sources that somewhat resemble a PT source. the classifier operates very well. However, for memoryless and other sources, it does not. Since the underlying assumption of the shaper is to observe a PT stream, this is not surprising. It is difficult to classify a stream as PT if it does not have any characteristics of a PT stream. Nonetheless, the next chapter introduces a few ideas

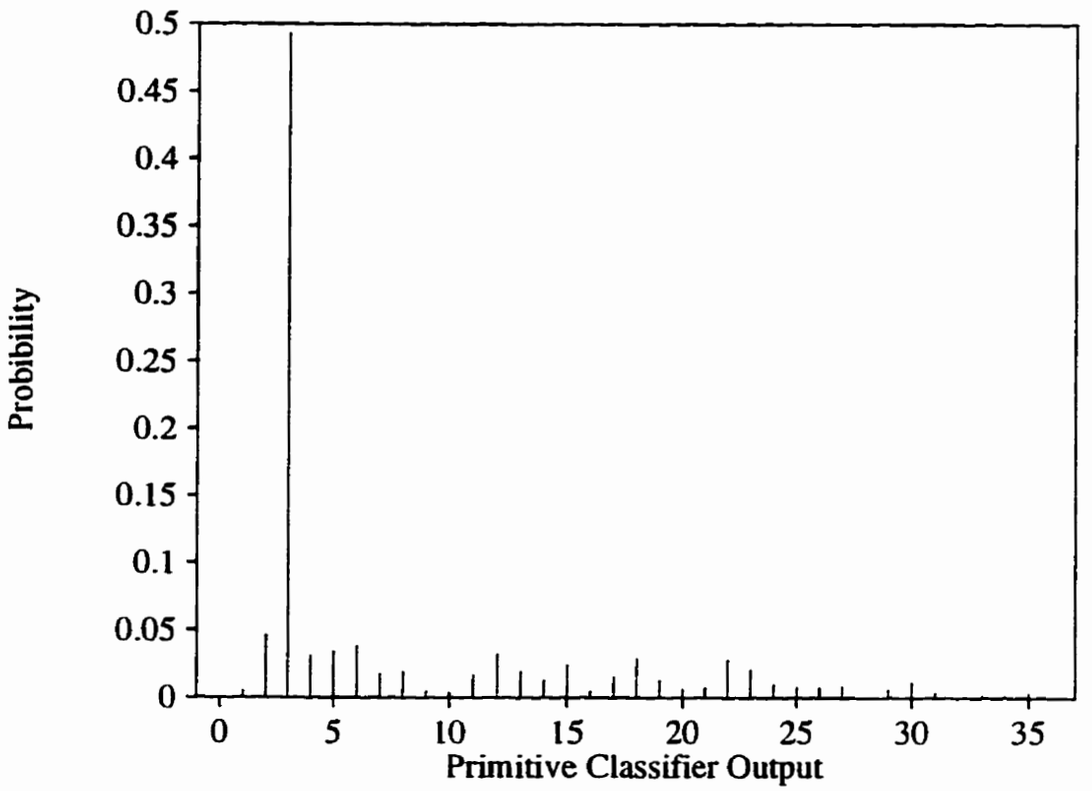


Figure 2.35: Source: MMBP. Histogram of Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

that can use the output of the traffic shaper to attempt to classify these streams.

Chapter 3

Traffic Classifier Applications

This chapter introduces two novel applications of traffic classification: Traffic Primitive Histogram Identification (TPHI) and Stream Transition Tracking (STT). Both of these stem from the observations made in Sections 2.3 and 2.4. TPHI is a method by which traffic classifications are made, and stored in a histogram. In this way, a source can be characterized, not by its cell statistics, but rather by its traffic primitive statistics. The idea is that if the neural network of the primitive classifier is trained properly, then it is able to discern patterns in a traffic stream that would not be readily apparent to a more conventional method, or simply casual observation. Hence, allowing the neural network to detect features, which may correspond to high-order statistics which are difficult to calculate, and then keeping a histogram of these features allows the characterization of unknown sources. As mentioned in Sections 1.3 and 1.4, this is one of the key motivations in ATM networks. The TPHI method is described in Section 3.1.

In Section 3.2, the STT method is introduced. This is a direct result of training

the neural network to recognize CBR-RC primitives. This method observes a cell stream and awaits changes that may occur. Depending on the type of change and its duration, certain actions can be taken.

Both of these applications of traffic classification provide information about traffic streams. As such, they can be of use to the elements of the UNI, namely CAC, UPC and traffic shaping.

3.1 Traffic Primitive Histogram Identification

There are two methods in which the TPHI scheme can operate: off-line and on-line. Each has advantages and disadvantages. First, since both methods function in much the same way, the concept of traffic primitive histograms is introduced.

Histograms of traffic primitives have already been presented in Section 2.4. The reader is encouraged to refer to the figures of that section, however, Figure 2.31 is reproduced here as Figure 3.1. Recall that Figure 3.1 shows the histogram of

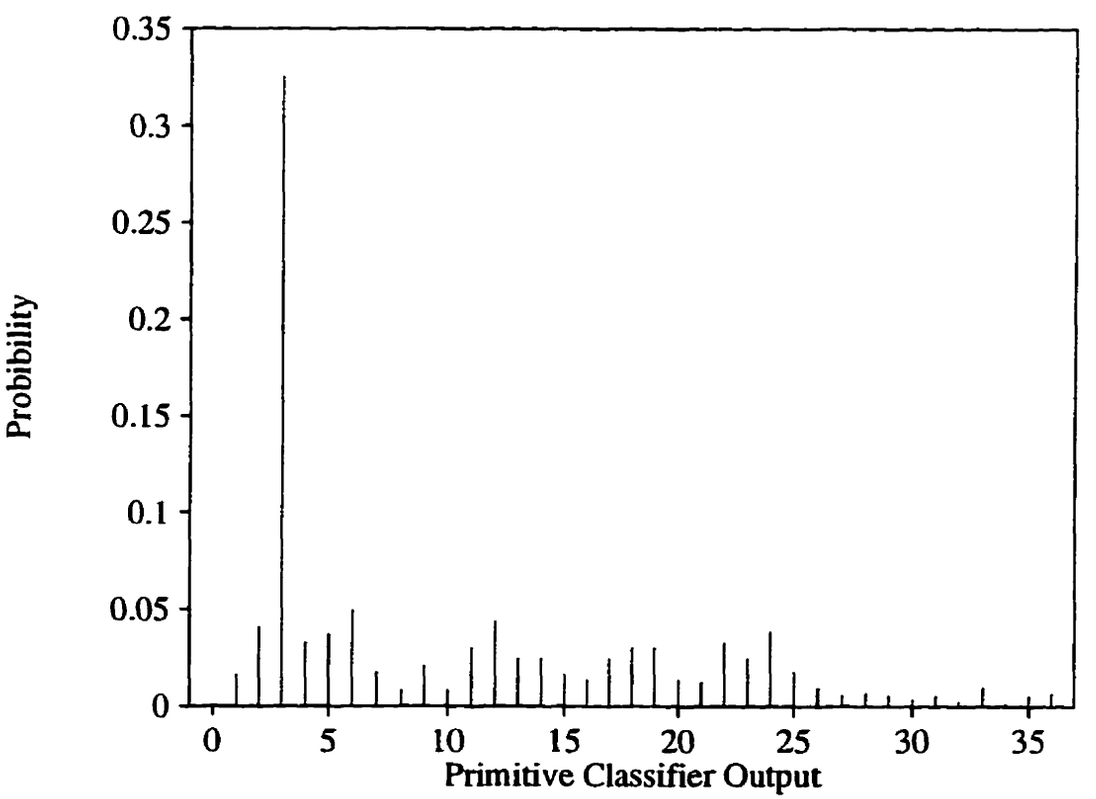


Figure 3.1: Source: $\mathcal{G}(0.5)$. Histogram of Classification Results using the 10–35–35–9 Neural Network based Primitive Classifier

primitive classifications of a source with interarrival times geometrically distributed, with mean 2 cells. In the last chapter, PT sources and sources which behave like packet trains are identified directly by the shaper. However, other sources like $\mathcal{G}(0.5)$ which are not encompassed in the traffic primitives require either additional classification, as mentioned in Section 2.1.6, or a method such as TPHI to help identify them.

Thus, with the ability to obtain traffic primitive histograms, this method is quite simple. As traffic classification is performed on a given cell stream, a histogram of these observations is kept. Then, from time to time, this histogram is compared with a "library" of histograms that the network provider must develop. This library is built up from previously known sources, whose characteristics have been identified. For example, an MMBP source is well known. However, to characterize an unknown MMBP source is not trivial. Worse still is attempting to identify that the source is MMBP to begin with. However, with TPHI this problem is simplified.

Two methods that can be used to decide whether the primitive histogram of a source in question matches one of the known sources in the library is to use a sectioning method, or a correlation method. Referring to Figure 3.1, it can be seen that the distribution of CBR-RC primitives observed takes on a definite periodic pattern. This can be observed in all the histograms of Section 2.4, and is most likely due to the priority assigned to individual traffic primitives as the training vectors are generated, as discussed in Section 2.2.1. Unfortunately, a direct relationship has proven to be elusive. Nevertheless, if each of these groupings of primitives is considered to be in a given section, then a simple method is to compare the total

probability in each section of an unknown source to the corresponding section of a known source. If the values agree within some threshold, then the sources are considered to be of the same class or type.

The correlation method entails simply multiplying the probability mass of each traffic primitive histogram observation of a known source with that of the source in question, and then summing these products to produce a measure. This measure is then compared to a benchmark measure obtained by performing the correlation of the library histogram with itself. If the correlation measure obtained is within some threshold value of the benchmark measure, then the source in question is considered to be of the same type as the library source.

This method can be performed both off-line and on-line. In the off-line method, users or the network provider can run trials on new or poorly characterized sources, and compare the results with the library histograms. In this way, new sources can be identified, and the library of known source histograms can expand. The drawback, of course, is that a user may still violate a Traffic Contract even though source identification has taken place.

In the on-line method, traffic streams are continuously or periodically characterized, to ensure that they are still within the same traffic class. This has the advantage of providing timely and up-to-date information to the UNI, however at the cost of increased complexity. Since the primitive classification utilizes neural networks, at least the updating of the traffic primitive histogram can be considered to be negligible. It is the histogram comparison with library histograms at which a bottleneck can form.

At this point it should be noted that work has already been performed on another histogram-based method [SSD93]. This scheme proposes a model for video traffic which is independent of video type and coding. A bit rate histogram is built, conditioned upon which video source in a given set is present at an ATM node. The assumption is also made that during any given frame of a video source, the cell arrival process is approximately Poisson. The main thrust of this paper is to use these histograms to model video traffic, which is then used to predict buffer occupancy and system performance at the ATM node. In addition, the video traffic studied is an aggregate of individual video sources. These two facts make comparison of the results given in the paper and the TPHI method difficult. In addition, the goals of the cited work and TPHI differ in that the TPHI method is applicable to, ideally, all traffic sources, not just video sources. Nevertheless, it may be possible to improve on the results of this work if, instead of employing the histogram method cited, the TPHI method based on traffic primitives is used. This could allow much of the node performance analysis to be applied to more general traffic sources.

As a final note, since the Traffic Primitive Histogram Identification method can be used to characterize sources, it can be used by every element of the UNI, including shaping. In fact, one of the shaping methods proposed in Chapter 4 relies on the fact that the source type is known and characterized. The next application of primitive classification does not characterize sources per se, instead it is more useful in monitoring their actions, and thus can be employed by UPC.

3.2 Stream Transition Tracking

From the discussion of the design of the traffic primitives in Section 2.1.3, especially the CBR-RC primitives, and from their properties as shown in the validation of the CBR and CBR-RC primitives of Sections 2.3.1 and 2.3.2, it can be seen that they are well suited for observing when a cell stream changes its character. For example, in Section 2.3.1, when a CBR source starts transmission after an initial thirty slots. it could be considered that the source changed its characteristics from an idle source. DG, to a CBR source.

As described, a sequence of classifications, usually CBR-RC, are emitted by the primitive classifier during the transition period. The idea of the STT is to monitor the classification sequence of a cell stream, and if a particular pattern is observed, then take some action. For example, Table 3.1 contains an example of the primitive sequence that occurs when a CBR source changes its rate from $\frac{C}{5}$ to $\frac{C}{2}$, which in general would have a detrimental effect on other users, and hence source policing should be performed. Of course, more complicated transitions can be allowed, such as the transition from a CBR to PT source, or the transition that occurs when an MMBP source transmits above its average rate.

However, due to the number of incorrect classifications observed in Sections 2.3 and 2.4, the STT requires some hysteresis to avoid making incorrect decisions as to whether a transition has occurred. Unfortunately, the specification of this can only be accomplished through the experience gained in operating the system.

One may wonder if a transition sequence is unique. Due to the way in which the training vectors are designed, after some thought the reader should rest assured

Table 3.1: CBR-RC Primitives Produced by a Rate Change from $\frac{C}{5}$ to $\frac{C}{2}$ by a CBR Source

Slot Number	Primitive Classification
1	CBR
2	CBR
3	CBR
4	RC $\frac{C}{4} \rightarrow \frac{C}{2}$
5	RC $\frac{C}{4} \rightarrow \frac{C}{2}$
6	RC $\frac{C}{4} \rightarrow \frac{C}{2}$
7	RC $\frac{C}{4} \rightarrow \frac{C}{2}$
8	RC $\frac{C}{3} \rightarrow \frac{C}{2}$
9	RC $\frac{C}{3} \rightarrow \frac{C}{2}$
10	CBR

that the transition sequences are in fact unique. If they were not, then a one to many neural network training vector mapping would exist. However, the training vectors are designed so that this cannot happen, and thus all transitions are unique.

Hence, the STT method could be employed by a smart policer to determine that a change has occurred in the traffic stream, and what that change implies. For example, if the source increases its rate by a small fraction, these cells can be tagged. However, if the rate increase is substantial, then these cells can be dropped.

This concludes this short chapter on the applications of traffic shaping. The next chapter introduces two shapers in detail, the Minimized Variance shaper which utilizes source characteristics in order to improve shaping, and the Burst-oriented shaper, which has the ability to unshape the shaped traffic stream at the destination UNI.

Chapter 4

Traffic Shaping

This chapter presents two shaping algorithms. As discussed in Section 1.3, Call Admission Control and Usage Parameter Control which occur at the User-Network Interface can be simplified if traffic shaping is performed. In addition, it is expected that network efficiency will increase, since cell scheduling at the multiplexer of ATM switches in the network can also be simplified.

First, the Variance Minimized shaper (MVS) is introduced in Section 4.1. This shaper attempts to minimize the interdeparture time variance of cells exiting the shaper, in an attempt to approximate the Ideal Shaper discussed in Section 1.3.2. If the interdeparture time variance is reduced to zero, then ideal shaping results. The approach taken to achieve this goal is to use knowledge specific to a given source. Therefore, the source type or traffic class must be known before shaping can proceed. Fortunately, Chapter 2 presented a method to accomplish this. The MVS could be thought of as another application of traffic classification and included with those presented in Chapter 3, however the importance of traffic shaping in

ATM networks warrants this special attention and hence this chapter. After the continuous and discrete time MVS models are presented in Sections 4.1.1 and 4.1.2, some results of shaping a few traffic classes are presented and discussed in Section 4.1.3. Aspects of this work are reported in [LM94].

Second, the Burst-oriented shaper (BOS) is introduced in Section 4.2. Like the MVS, and as mentioned in Section 1.4, the goal of the BOS is to reduce the interdeparture time variance of cells leaving the shaper to zero. Unlike the MVS, the BOS does not require knowledge of the traffic type. In this way, it is similar to some of the shapers in the literature, cited in Section 1.2.3.3. However, the novel approach used here stems from the insights gained in the development of traffic classification of Chapter 2 which lead to the contribution of this shaper, the concept of Ideal Unshaping, as discussed in Section 1.3.2. In addition to attempting to generate a deterministic traffic stream at its output, the BOS also provides information embedded in the shaped cell stream to an unshaper at the destination UNI, as depicted in Figure 1.3. In this way, a stream with the exact same characteristics is presented to the end user as the one which entered the source UNI. This can be very important to certain sources which are sensitive to delays within their cell stream caused by either shaping or network congestion. This information could also be of value to intermediate network nodes, since it characterizes the traffic stream. The BOS model is presented in Section 4.2.1, and the shaping and unshaping algorithms in Sections 4.2.2 and 4.2.3, respectfully. Finally, the results of shaping a few traffic types are provided in Section 4.2.5.

It should be reiterated at this point that unlike most shapers in the literature,

the two to be discussed attempt to create deterministic traffic streams from probabilistic streams. This, as mentioned, is beneficial at the UNI. In addition, if all traffic streams flowing through an ATM network were deterministic, deterministic scheduling and multiplexing at the switches would result, and so congestion would be avoided. It is acknowledged, however, that due to delay and other constraints, that some sources cannot be shaped.

4.1 Minimized Variance Shaper

The proposed cell-space shaper is modeled as a FIFO[1] queue and server. The service time is dependent on the cell interarrival times of the arrival process of a given traffic stream, and on whether the shaper system is empty upon a cell arrival. The service time is optimized, assuming the shaper is empty, so that the interdeparture time variance of cells leaving the shaper is minimized. In order to limit the size of the shaper queue, a heuristic is included in the optimization.

The shaping algorithms of [Bro92, Cha91] are somewhat similar to that presented here; however, in this work it is assumed that the shaper has knowledge of the type of traffic presented to it, so that the shaping can be tailored to the characteristics of a particular cell stream. This information can be obtained via the traffic classification method of Chapter 2. Note that while the MVS is presented in the context of ATM networks, it is applicable to any packet switching network.

In order to shape sources where the cell arrival process is known and is analytically tractable, a continuous time shaper model is discussed in Section 4.1.1. On the other hand, for cases when the cell arrival process is intractable, or only a cell interarrival histogram is available, a discrete time version of the shaper model is discussed in Section 4.1.2. To complete the study of the MVS, some examples of its operation appear in Section 4.1.3.

[1] First in, first out.

4.1.1 Continuous Time Shaper Model

Consider a cell-space shaping device that delays a cell if its interarrival time,[2] $I_i \in \mathbb{R}^+$, is below a certain threshold, and allows the cell to pass if its interarrival time is above the threshold; call this threshold the shaping parameter K. Also, denote the interdeparture time[3] of a cell leaving the shaper as D_i. The problem is how to choose K such that the cell interdeparture time variance of a given traffic stream is minimized.

The shaper can be represented as a queue and server, as shown in Figure 4.1. From the figure, it can be seen that $D_i = I_i + \mathcal{D}(I_i)$, where $\mathcal{D}(I_i)$ represents the "service time" or delay as a cell passes through the shaper. In general, the traffic shaper implements the delay function

$$\mathcal{D}(I_i) = \begin{cases} K - I_i & \text{if } I_i \leq K \\ 0 & \text{if } I_i > K, \end{cases} \tag{4.1}$$

where K is yet to be determined. Ideally, if a cell is delayed no further arrivals occur until that cell has left the server; in other words, no queueing takes place. But very few, if any, sources behave in this manner. Due to the fact that this service time is dependent on the cell interarrival time, methods utilizing embedded Markov chains cannot be applied [Kle75, Wol89]. Based on the short term arrival characteristics of the cell traffic, the shaping can be either "soft" or "hard," as described below.

---

[2]The cell interarrival time is defined as the amount of time that has elapsed between the arrival time to the shaper of the cell in question and that of the previous cell arrival.

[3]The cell interdeparture time is analogous to the cell interarrival time, except cells are leaving the shaper instead of arriving.

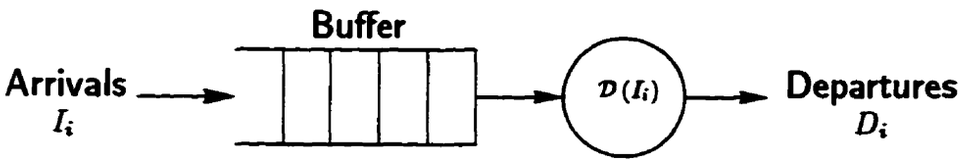


Figure 4.1: Continuous Time Shaper Model

If cell interarrival times are large relative to the shaping parameter, that is $I_i \gg K$, little or no shaping takes place. In this case, the shaping can be described as being soft, that is the probability that the shaper delays a cell is low. In this case, $\mathcal{D}(I_i) = 0 \ \forall \ i$. At the other extreme, if cell interarrival times are small relative to the shaping parameter, that is $I_i \ll K$, hard shaping is said to take place. The probability that the shaper delays a cell is high, since K "encompasses" most interarrival times. In this case, $\mathcal{D}(I_i) = K \ \ \forall \ i$. This, unfortunately, implies an infinite shaper queue if cell arrivals occur while previously arrived cells are being delayed.

Hence, $\mathcal{D}(I_i)$ can be thought of as an "asymptotic" value of delay as the shaper changes its characteristics between soft shaping and hard shaping. As shaping becomes soft, the "trivial" solution for the shaping parameter results in $\mathcal{D}(I_i) = 0$, in which case shaping no longer occurs. The interdeparture time variance is the same as the interarrival time variance, with no queueing at the shaper. As the shaping becomes hard, the trivial solution is $\mathcal{D}(I_i) = \infty$. This has the effect of reducing the interdeparture time variance to zero, but it requires an infinite length queue.

4.1.1.1 The Continuous Time MVS Algorithm

In order to bring these two conflicting solutions together, a compromise must be made between hard and soft shaping. Thus, if the shaper is empty, cells are delayed an amount of time equal to $\mathcal{D}(I_i)$ given in Equation (4.1). If a cell arrives at the shaper to find another cell being delayed, or cells queued, it is enqueued and its delay set to $\mathcal{D}(I_i) = K$. The delay function now becomes

$$\mathcal{D}(I_i) = \begin{cases} \begin{cases} K - I_i & \text{if } I_i \leq K \\ 0 & \text{if } I_i > K \end{cases} & \text{if the shaper is empty,} \\ K & \text{if the shaper is non-empty.} \end{cases} \tag{4.2}$$

As a result, when cells arrive with intervening gaps the shaper operates in the soft mode, attempting to make $D_i = K$, and when cells arrive in bunches the shaper operates in the hard mode with $D_i = K$. The underlying assumption is that cells which arrive during hard shaping are "spread" into areas of the cell stream which *would have* been shaped in the soft shaping mode. This may cause an increase in queue length, and so the reason for the compromise: it is necessary to develop a strategy to bound the queue growth. One such strategy is to introduce a heuristic to allow the user to specify a maximum allowable increase in mean interdeparture time. All that remains is to determine the value of K.

4.1.1.2 Determination of K

Consider a user traffic stream which can be characterized by a cell interarrival time process with probability density function (pdf) $i(t)$ and cumulative distribution

function (CDF)[4] $I(t)$. After passing through the shaper, the cell interdeparture time variance V_{D_i} is given by

$$\begin{aligned} V_{D_i} &= E\left[(D_i - E[D_i])^2\right] \\ &= E\left[D_i^2\right] - E[D_i]^2 \\ &= \int_0^K K^2 i(t)\, dt + \int_K^\infty t^2 i(t)\, dt - \left[\int_0^K Ki(t)\, dt + \int_K^\infty ti(t)\, dt\right]^2 \\ &= K^2 Pr\{I \le K\} + \int_K^\infty t^2 i(t)\, dt - \left[KPr\{I \le K\} + \int_K^\infty ti(t)\, dt\right],^2 \end{aligned} \tag{4.3}$$

assuming that cells always find the shaper empty; in other words, soft shaping mode. in which no queueing takes place. The implicit error of this assumption has to do with the $\int_K^\infty ti(t)\, dt$ and $\int_K^\infty t^2 i(t)\, dt$ terms in Equation (4.3). These terms deal with the interarrival times of cells such that $I_i > K$, but the shaper is operating in *soft mode* instead of hard mode, by the above assumption. Since some of these cells encounter a non-empty shaper queue, they are, in effect, hard shaped. Thus, these terms in Equation (4.3) are too large and the terms $\int_0^K K^2 i(t)\, dt$ and $\int_0^K Ki(t)\, dt$ are too small. Hence, this expression for the interdeparture time variance from the shaper is an upper bound.

Now consider an optimization problem wherein the cell interdeparture time variance V_{D_i} of the stream leaving the shaper is to be minimized, subject to the constraint that the mean interdeparture time, $\overline{D_i}$, is to be less than some multiple

[4]For definitions of the pdf and CDF, refer to [Pap84].

γ of the cell mean interarrival time, $\frac{1}{\lambda}$. The optimization problem can be stated as

$$\min_{K \geq 0} \{V_{D_i}\} \quad \text{such that} \quad \overline{D_i} \leq \gamma \frac{1}{\lambda}, \tag{4.4}$$

The solution of this problem is dependent upon the characterization of the traffic stream. In the following section, this is performed for a Poisson source, and for a on-off source in Section 4.1.1.4.

4.1.1.3 Minimization of V_{D_i} for a Poisson Source

If the traffic source is Poisson, then cell interarrival times are exponentially distributed with

$$i(t) = \lambda e^{-\lambda t} \qquad \text{(pdf)} \tag{4.5}$$

$$I(t) = 1 - e^{-\lambda t} \qquad \text{(CDF)}, \tag{4.6}$$

for $0 \leq t \leq \infty$. After shaping, the interdeparture time variance V_{D_i} is given by the following equation, which is obtained by substituting Equations (4.5) and (4.6) into Equation (4.3),

$$\begin{aligned} V_{D_i} &= K^2 Pr\{I \leq K\} + \int_K^\infty t^2 \lambda e^{-\lambda t}\, dt - \left[K Pr\{I \leq K\} + \int_K^\infty t \lambda e^{-\lambda t}\, dt\right]^2 \\ &= K^2 \left(1 - e^{-K}\right) + K^2 e^{-\lambda K} + \frac{2K}{\lambda} e^{-\lambda K} + \frac{2}{\lambda^2} e^{-\lambda K} \\ &\quad - \left[K\left(1 - e^{-K}\right) + K e^{-\lambda K} + \frac{1}{\lambda} e^{-\lambda K}\right]^2 \\ &= \frac{2}{\lambda^2} e^{-\lambda K} - \frac{1}{\lambda^2} e^{-2\lambda K}. \end{aligned} \tag{4.7}$$

In order to allow the specification of a maximum tolerable delay added by shaping, the mean interdeparture time $\overline{D_i}$ of cells leaving the shaper can be bounded by, say $\gamma\frac{1}{\lambda}$, where

$$\begin{aligned}\overline{D_i} &= \int_0^K Ki(t)\,dt + \int_K^\infty ti(t)\,dt \\ &= KPr\{I_i \le K\} + \int_K^\infty ti(t)\,dt \\ &= K + \frac{1}{\lambda}e^{-\lambda K}.\end{aligned} \tag{4.8}$$

Therefore the problem of Equation (4.4) can be set up as a constraint optimization with objective function

$$L = V_{D_i} + \mu\left(\overline{D_i} - \gamma\frac{1}{\lambda}\right), \tag{4.9}$$

where μ is an arbitrary Lagrangian multiplier. Taking the first order partial derivatives gives

$$\frac{\partial L}{\partial \mu} = \overline{D_i} - \gamma\frac{1}{\lambda} \tag{4.10}$$

$$\frac{\partial L}{\partial K} = \left(\frac{-2}{\lambda}e^{-\lambda K} + \mu\right)\left(1 - e^{-\lambda K}\right). \tag{4.11}$$

Setting Equation (4.10) to zero yields

$$K + \frac{1}{\lambda}e^{-\lambda K} - \gamma\frac{1}{\lambda} = 0, \tag{4.12}$$

which is the solution for the shaping parameter K. Bear in mind that Equation (4.12) requires a numerical solution. Setting Equation (4.11) to zero results in $\mu = \frac{2}{\lambda}e^{-\lambda K}$ and $K \neq 0$. Observe that the constraint will be active only if $\mu > 0$, which holds true for all K. Hence the constraint is always active and the solution of Equation (4.12) minimizes V_{D_i}. Notice that at $K = \infty$, $\mu = 0$, so the constraint becomes inactive. This solution is in agreement with the intuition developed under hard shaping.

4.1.1.4 Minimization of V_{D_i} for an On-off Source

The on-off traffic source described here is the same as that presented in [SW86]. It can be characterized by[5]

$$i(t) = (1 - \alpha\Delta)\,\delta(t - \Delta) + \alpha\Delta\beta e^{-\beta(t-\Delta)}\,U(t - \Delta) \quad \text{(pdf)} \tag{4.13}$$

$$I(t) = \begin{cases} 0 & \text{for } 0 < t < \Delta \\ 1 - \alpha\Delta e^{-\beta(t-\Delta)} & \text{for } \Delta \leq t \leq \infty \end{cases} \quad \text{(CDF)} \tag{4.14}$$

where α is the mean length of the geometrically distributed on period, β is the mean length of the exponentially distributed off period, and Δ is the (deterministic) time between cell arrivals in the on state.

After shaping, the interdeparture time variance is given by

$$V_{D_i} = E\left[(D_i - E[D_i])^2\right]$$

[5]Note that $\delta(\cdot)$ represents the impulse function, and $U(\cdot)$ represents the unit step function. For definitions, refer to [Pap80].

$$= \int_{\Delta}^{K} K^2 i(t)\, dt + \int_{K}^{\infty} t^2 i(t)\, dt - \left[\int_{\Delta}^{K} K i(t)\, dt + \int_{K}^{\infty} t i(t)\, dt\right]^2 . \tag{4.15}$$

The lower limit of the integrals is Δ since $K < \Delta$ implies that no shaping takes place. Evaluating,

$$\begin{aligned} V_{D_i} &= K^2 Pr\{I \le K\} + \int_{K}^{\infty} t^2 \alpha\Delta e^{-\beta(t-\Delta)}\, dt \\ &\quad - \left[K Pr\{I \le K\} + \int_{K}^{\infty} t\alpha\Delta e^{-\beta(t-\Delta)}\, dt\right]^2 \\ &= K^2\left\{1 - \alpha\Delta e^{-\beta(K-\Delta)}\right\} + \alpha\Delta e^{-\beta(K-\Delta)}\left\{K^2 + \frac{2K}{\beta} + \frac{2}{\beta^2}\right\} \\ &\quad - \left[K\left\{1 - \alpha\Delta e^{-\beta(K-\Delta)}\right\} + \alpha\Delta e^{-\beta(K-\Delta)}\left\{K + \frac{1}{\beta}\right\}\right]^2 \\ &= \frac{\alpha\Delta}{\beta} e^{-\beta(K-\Delta)}\left(2 - \alpha\Delta e^{-\beta(K-\Delta)}\right), \end{aligned} \tag{4.16}$$

and, as well,

$$\begin{aligned} \overline{D_i} &= \int_{\Delta}^{K} K i(t)\, dt + \int_{K}^{\infty} t i(t)\, dt \\ &= K Pr\{I \le K\} + \alpha\Delta \int_{K}^{\infty} t\beta e^{-\beta(t-\Delta)}\, dt \\ &= K\left\{1 - \alpha\Delta e^{-\beta(K-\Delta)}\right\} + \alpha\Delta e^{-\beta(K-\Delta)}\left\{K + \frac{1}{\beta}\right\} \\ &= K + \frac{\alpha\Delta}{\beta} e^{-\beta(K-\Delta)}. \end{aligned} \tag{4.17}$$

Solving Equation (4.4), as in section 4.1.1.3 by taking derivatives of the objective

function, Equation (4.9), produces

$$\frac{\partial L}{\partial \mu} = \overline{D_i} - \gamma \frac{1}{\lambda} \tag{4.18}$$

$$\frac{\partial L}{\partial K} = \left(-2\frac{\alpha\Delta}{\beta} e^{-\beta(K-\Delta)} + \mu \right) \left(1 - \alpha\Delta e^{-\beta(K-\Delta)} \right). \tag{4.19}$$

Setting Equations (4.18) and (4.19) to zero yields

$$K + \frac{\alpha\Delta}{\beta} e^{-\beta(K-\Delta)} - \gamma \frac{1}{\lambda} = 0, \tag{4.20}$$

which upon numerical solution, gives the optimum value for K. The constraint is active for $\Delta \leq K < \infty$. With an on-off source, recognize that $\lambda = \frac{1}{\Delta + \frac{\alpha\Delta}{\beta}}$.

4.1.2 Discrete Time Shaper Model

The discrete time model is analogous to the continuous time model of Section 4.1.1. However, in this case the pdf of the cell interarrival distribution need not be continuous nor analytically tractable; in fact, the network provider can require only a cell interarrival histogram. This is useful for sources where the arrival distributions are either difficult to obtain or unknown, such as those anticipated for future applications.

Consider a cell generating process in which cell interarrival time is the random variable, $I_N \in \mathbb{R}^+$, where the subscript N emphasizes the fact that the interarrival time can take on only N discrete values. Thus, a given realization of an interarrival

time, $I_i(N)$, is taken from the interarrival time set

$$\left\{ I_i(N) \mid I_i(N) \in \mathbb{R}^+,\ 1 \leq i \leq N,\ \text{and } I_i(N) < I_j(N),\ i < j \right\}. \tag{4.21}$$

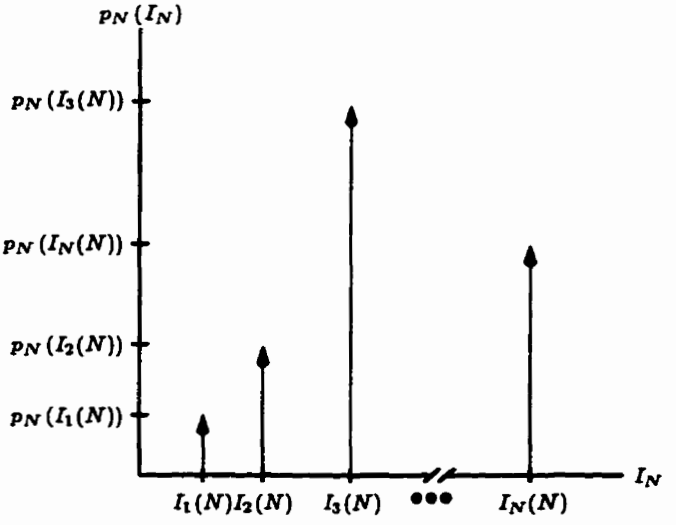
Note that the interarrival times are ordered in increasing value, with $I_1(N)$ the smallest in value, and $I_N(N)$ the largest. If $\mathcal{I}_N(I_N)$ is taken to represent the CDF of the random variable I_N, and if $p_N(I_i(N))$ represents the probability that the interarrival time $I_i(N)$ occurs, then

$$\begin{aligned} p_N(I_i(N)) &= Pr\{I_N = I_i(N)\} \\ &= Pr\{I_N \leq I_i(N)\} - Pr\{I_N \leq I_{i-1}(N)\} \\ &= \mathcal{I}_N(I_i(N)) - \mathcal{I}_N(I_{i-1}(N)). \end{aligned} \tag{4.22}$$

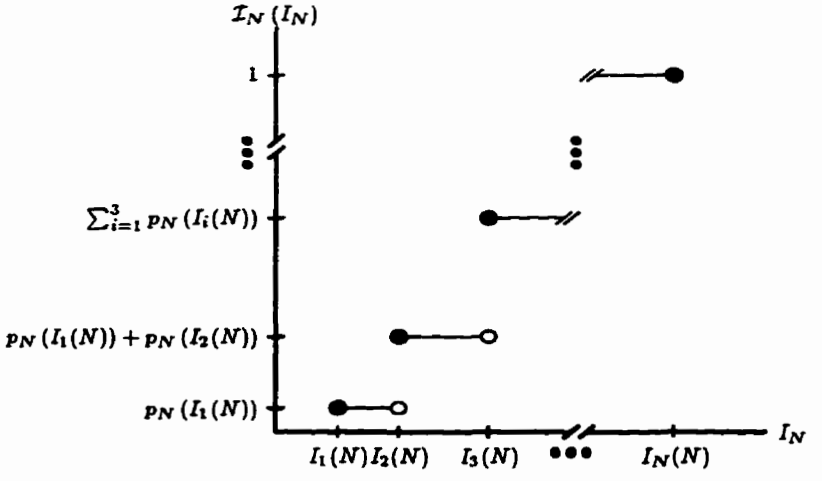
Thus the $p_N(I_i(N))$, $i = 1, 2, 3, \ldots, N$, specify the probability distribution[6] of I_N. Figures 4.2(a) and 4.2(b) illustrate this. In order to provide a point of comparison between the continuous time model of Section 4.1.1 and the discrete time model of this section, consider a source with exponential interarrival times, such as that of Equations (4.5) and (4.6). If the CDF of Equation (4.6) is partitioned into m "sections," that is discretized as shown in Figure 4.3, this would give rise to $m + 1$ interarrival times, $I_1(N)$, $I_2(N) \ldots, I_{m+1}(N)$, where $I_{m+1}(N) \equiv I_N(N)$.

The reader should note that most continuous interarrival distributions are defined for all positive times, that is $I_i \in \mathbb{R}^+$. However, from Equation (4.21), this

[6]See [Pap84].



(a) Probability Distribution, $p_N(\mathcal{I}_N)$, of the Discrete Random Variable I_N



(b) Cumulative Distribution Function (CDF), $\mathcal{I}_N(I_N)$ of the Discrete Random Variable I_N

Figure 4.2: Probability Distribution and Cumulative Distribution Function (CDF) for the Interarrival Time Random Variable I_N.

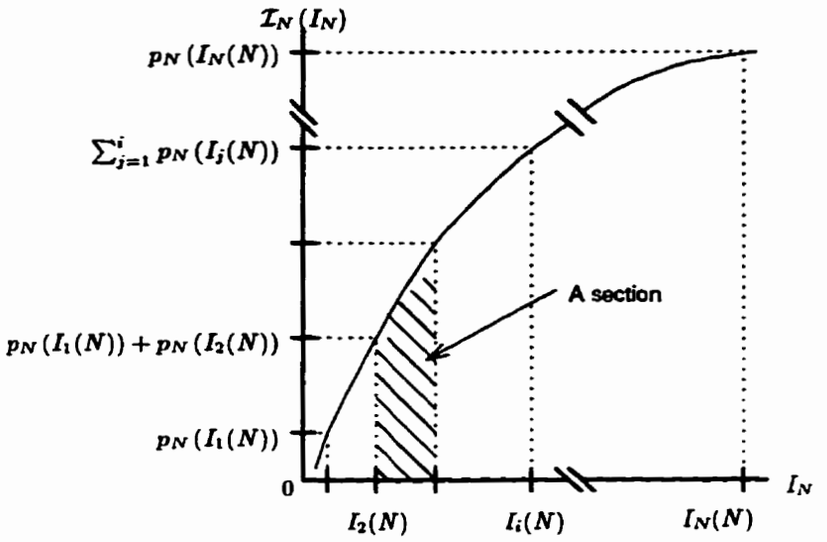


Figure 4.3: A Discretized Exponential CDF

discrete interarrival time distribution has a clearly defined maximum value, namely $I_N(N)$. Thus, in order to more accurately compare the results of the discrete model and continuous model shapers, the continuous exponential source needs to be "truncated" at a suitable value. For example, if ξ is defined to be the *truncation factor* in terms of probability, then the maximum value of a truncated CDF is $I_N(N)$, that is $Pr\{I_N \leq I_N(N)\} = 1 - \xi$. If $\hat{i}(t)$ represents the truncated version of the pdf $i(t)$, then in order to keep the total probability in a truncated distribution unity, that is $\int_0^{I_N(N)} \hat{i}(t)\, dt = 1$, $i(t)$ must be scaled. Thus, the truncated versions of Equations (4.5) and (4.6) are

$$\hat{i}(t) = \frac{\lambda e^{-\lambda t}}{1 - \xi} \qquad \text{(pdf)} \tag{4.23}$$

$$\hat{I}(t) = \frac{1-e^{-\lambda t}}{1-\xi} \qquad \text{(CDF)}. \tag{4.24}$$

The untruncated CDF $I(t)$ and maximum probability of $1-\xi$ can be employed to obtain an expression for the maximum value of the truncated distribution,

$$\begin{aligned} Pr\{t \leq I_N(N)\} &= 1-\xi \\ 1-e^{-\lambda I_N(N)} &= 1-\xi \\ -\lambda I_N(N) &= \ln(\xi), \end{aligned}$$

and so

$$I_N(N) = \frac{1}{\lambda}\ln\left(\frac{1}{\xi}\right). \tag{4.25}$$

Further, assume that the m sections of Figure 4.3 occur with equal probability, except for the last section which has ξ less probability, and so

$$p_N(I_i(N)) = \begin{cases} \frac{1}{N} & \text{for } i = 1,\ 2,\ \ldots,\ N-1, \\ \frac{1}{N}-\xi & \text{for } i = N, \end{cases} \tag{4.26}$$

In general, given a truncation factor ξ, truncated pdf and CDF $\hat{i}(t)$ and $\hat{I}(t)$, and an underlying continuous distribution divided into m sections of equal probability $p_N(I_i(N)) = \frac{1}{m}$, except for $p_N(I_N(N)) = \frac{1}{m}-\xi$, then the $N = m+1$ discrete interarrival times are given by $I_i(N) = \mathcal{I}_N^{-1}\left(\sum_{j=1}^{i} p_N(I_j(N))\right)$, $i = 1,\ 2,\ \ldots,\ N-1$, and $I_N(N) = \mathcal{I}_N^{-1}(1-\xi)$, where $\mathcal{I}_N^{-1}$ represents the inverse of $\hat{I}(t)$.

The cell spacing model is almost exactly the same as that for the continuous time sources of section 4.1.1, as Figure 4.4 shows. The only difference comes in a slight change in notation due to the "sectioning" of the source interarrivals. Hence, the discussion of section 4.1.1 applies equally well here. As with the continuous time case, treating the shaper as a queue and server, the discrete time equivalent of Equation (4.2) is given by

$$\mathcal{D}(I_i(N)) = \begin{cases} \begin{cases} K(N,m) - I_i(N) & \text{if } i = 1,\ 2,\ \ldots,\ m-1 \\ 0 & \text{if } i = m,\ m+1,\ \ldots,\ N \end{cases} & \text{if the shaper is empty,} \\ K(N,m) & \text{if the shaper is non-empty.} \end{cases} \tag{4.27}$$

In this case, $\mathcal{D}(I_i(N))$ is the discrete time equivalent of $\mathcal{D}(I_i)$, and $K(N,m)$ is the shaping parameter yet to be determined. In addition, be aware that $K(N,m)$ is dependent not only on the total number of sections N, but it will lie within one particular section, m, as well.

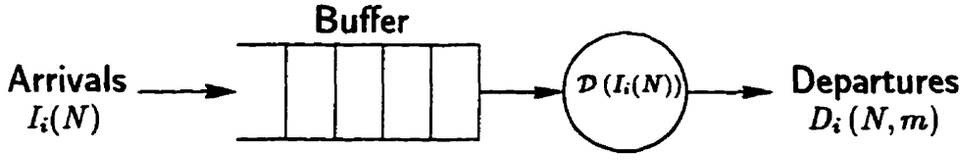


Figure 4.4: Discrete Time Shaper Model

4.1.2.1 Determination of $K(N,m)$

With the discrete time source characterized, it can be seen that after the cells pass through the shaper, the interdeparture time variance $V_{D_i(N,m)}$ is

$$V_{D_i(N,m)} = \sum_{i=1}^{m-1} p_i(N) K(N,m)^2 + \sum_{i=m}^{N} p_i(N) I_i(N)^2 - \left[\sum_{i=1}^{m-1} p_i(N) K(N,m) + \sum_{i=m}^{N} p_i(N) I_i(N) \right]^2, \tag{4.28}$$

assuming that cells always find the shaper empty, that is a soft shaper. In order to conveniently write these equations, define

$$a(N,m) \triangleq \sum_{i=1}^{m-1} p_i(N), \qquad b(N,m) \triangleq \sum_{i=1}^{m-1} p_i(N) I_i(N),$$
$$a'(N,m) \triangleq \sum_{i=m}^{N} p_i(N), \qquad b'(N,m) \triangleq \sum_{i=m}^{N} p_i(N) I_i(N),$$

and

$$c'(N,m) \triangleq \sum_{i=m}^{N} p_i(N) I_i(N)^2.$$

Then

$$V_{D_i(N,m)} = a(N,m)\, a'(N,m)\, K(N,m)^2 - 2a(N,m)\, b(N,m)\, K(N,m) + c'(N,m) - b'(N,m)^2. \tag{4.29}$$

Keeping in mind the same optimization problem as in the continuous time case.

where the mean interdeparture time $\overline{D_i(N,m)}$ is bound by $\gamma\frac{1}{\lambda}$, produces

$$\min_{K(N,m)\geq 0}\left\{V_{D_i(N,m)}\right\} \quad \text{such that} \quad \begin{cases} \overline{D_i(N,m)} \leq \gamma\frac{1}{\lambda} \\ I_{m-1}(N) \leq K(N,m) \leq I_m(N) \\ 2 \leq m \leq N, \end{cases} \tag{4.30}$$

which gives the optimum $K(N,m)$ for a given N, for all m. In addition. it can also be shown that

$$\overline{D_i(N,m)} = a(N,m)\,K(N,m) - b(N,m) + b'(N,m)\,. \tag{4.31}$$

Forming the Lagrange equation results in

$$L = V_{D_i(N,m)} + \mu_1\left(\overline{D_i(N,m)} - \gamma\frac{1}{\lambda}\right) + \mu_2\left(I_{m-1}(N) - K(N,m)\right) + \mu_3\left(K(N,m) - I_m(N)\right), \tag{4.32}$$

where μ_1, μ_2, μ_3 are arbitrary Lagrangian multipliers. Taking the partial derivatives results in

$$\frac{\partial L}{\partial \mu_1} = \overline{D_i(N,m)} - \gamma\frac{1}{\lambda} \tag{4.33}$$

$$\frac{\partial L}{\partial \mu_2} = I_{m-1}(N) - K(N,m) \tag{4.34}$$

$$\frac{\partial L}{\partial \mu_2} = K(N,m) - I_{m-1}(N) \tag{4.35}$$

$$\frac{\partial L}{\partial K(N,m)} = 2a(N,m)\,a'(N,m) - 2a(N,m)\,b'(N,m)$$

$$+ a(N,m)\,\mu_1 - \mu_2 + \mu_3. \tag{4.36}$$

Setting these four equations to zero and evaluating the cases which arise from the possible combinations of activity and inactivity of the three constraints results in the general solution for $K(N,m)$ as follows:

$$K(N,m) = \begin{cases} I_{m-1}(N) & \text{if } a(N,m)\left(a'(N,m)\,I_{m-1}(N) - b'(N,m)\right) \geq 0, \\ I_m(N) & \text{if } a(N,m)\left(b'(N,m) - a'(N,m)\,I_m(N)\right) \geq 0, \\ \frac{b(N,m)-b'(N,m)+\gamma\frac{1}{\lambda}}{a(N,m)} & \text{if } b'(N,m) - \frac{a'(N,m)\left(b(N,m)+\gamma\frac{1}{\lambda}\right)}{a(N,m)} \geq 0, \\ \frac{b'(N,m)}{a'(N,m)} & \text{otherwise.} \end{cases} \tag{4.37}$$

The conditions must be checked in the order given for the constraints to be satisfied.

Unlike the continuous time case, this is as far as the derivation can proceed for the discrete time model without being given the $p_i(N)$ and $I_i(N)$. The following section presents results of shaping a few example sources.

4.1.3 Results

This section gives the results of the Continuous Time MVS (CT-MVS) in some detail, since it is analytically tractable. In Section 4.1.3.1, sources that can be characterized by the exponential distribution are studied, however some comments are made about the on-off source introduced in Section 4.1.1.4 as well. Section 4.1.3.2 begins by comparing the results of the Discrete Time MVS (DT-MVS) operating on a discretized version of an exponential source presented in Section

4.1.3.1. Then, the more interesting cases of an MMBP, and PT sources follow.

A software suite, written in C, is utilized to calculate the optimum value of K and $K(N,m)$. Then, cell arrivals of the appropriate distribution are generated and the operation of the shaper is simulated. In all the following, ten runs are performed in order to obtain confidence intervals. However, in only the worst cases are the intervals as large as five percent of the value in question. In addition, the intervals do not increase. Hence, for clarity, confidence intervals are not included on the graphs that follow. In addition, there are 10,000 arrivals per simulation run.

The arrival rates are measured in cells per unit time. Since both MVS algorithms are concerned with interarrival time distributions and not actual interarrival times, any time scale suffices. Thus, an arrival rate of $\frac{1}{\lambda} = 1.0$ can be interpreted as, on average, one cell arrives during a "normalized" unit of time. An example unit could be one hundred ATM slots.

4.1.3.1 Continuous Time MVS

Figures 4.5–4.7 show how the shaper can effectively shape sources with exponentially distributed interarrival times. In Figure 4.5, the increase in mean delay is constrained to be one percent of the mean interarrival time, that is $\gamma\frac{1}{\lambda} = 1.01$, since $\frac{1}{\lambda} = 1.0$. This does not leave much room in which the shaper can operate, and as can be seen, only about 17% of the cell arrivals are shaped to the optimum value of the shaping parameter, $K = 0.145$ (time units). For the case where the mean delay is allowed to increase to fifteen percent shown in Figure 4.6, the shaper operates much better, with over 45% of the cell arrivals shaped. This means that almost half of the time this stream resembles a deterministic stream, with a constant interval

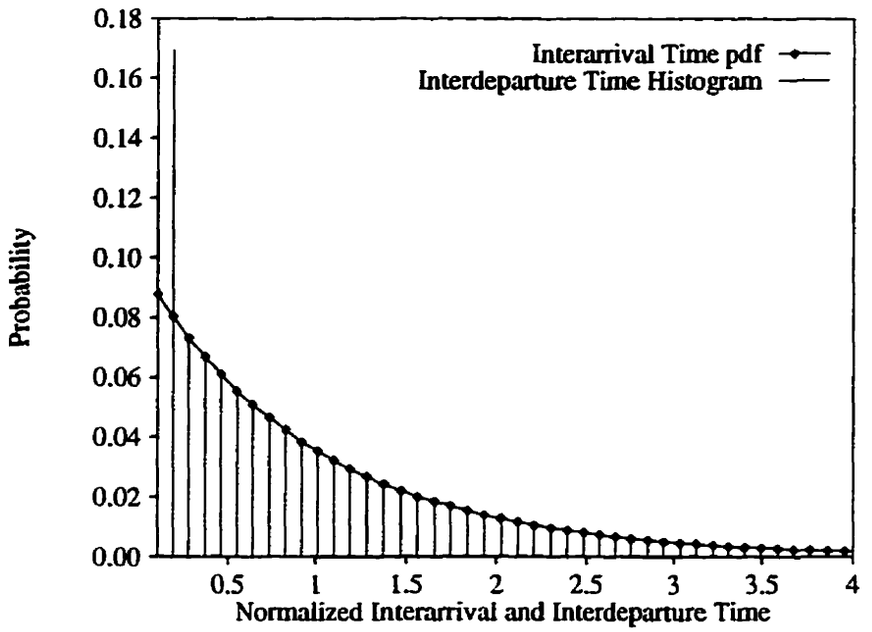


Figure 4.5: Poisson Source: $\frac{1}{\lambda} = 1.0$, $\gamma = 1.01$, $K = 0.145$

of $K = 0.602$. As can be seen from the figure, approximately 0.1% of the cell interdeparture times are less that K. These represent cells at the tail end of a long interarrival time in the cell stream which arrive at the shaper K time units or less after the shaper has become idle.

When cells are queued in the shaper during one of these long interarrival times, they depart spaced equally with interval K. This has the effect of spreading cells which arrive in bunches into the long interarrival times. However, for this small percentage, the interarrival time is just long enough so that the queue empties, and the shaper becomes idle. The cell arrival "just misses" encountering a non-empty shaper by an amount of time less than K. Since a cell that encounters an empty

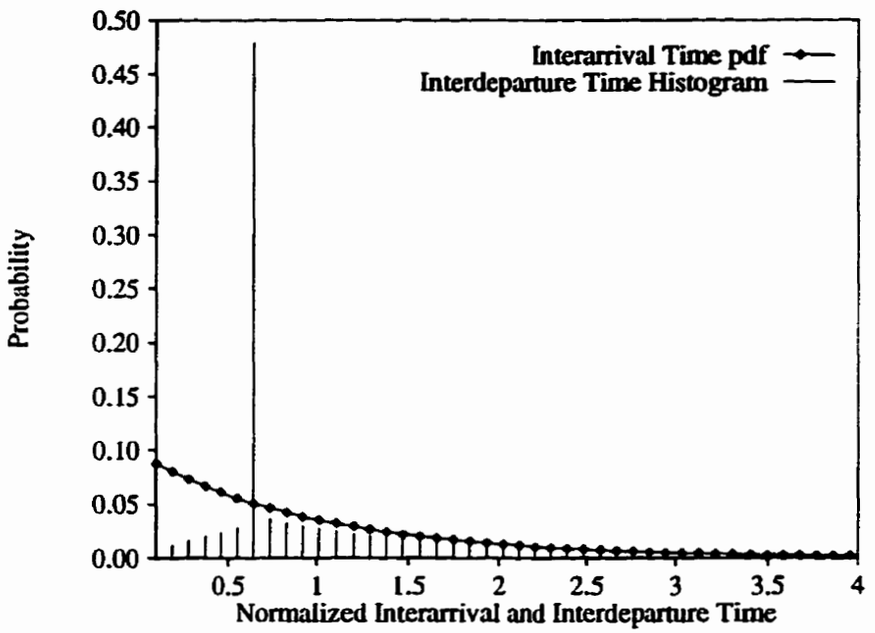


Figure 4.6: Poisson Source: $\lambda = 1.0$, $\gamma = 1.15$, $K = 0.602$

shaper with $I_i > K$ is not shaped, as Equation (4.2) dictates, the interdeparture time is this value less than K.

On the other hand, for the approximately 50% of the remaining interarrival times, the interarrival time is so long that it is much greater than K. Nonetheless. as shown by the interarrival times just to the right of K in Figure 4.6, since the interdeparture time histogram values are less than the unshaped interarrival time pdf values, some of the long interarrival times are shortened by the cell spreading mentioned above. In fact, the area between the interarrival time pdf and the interdeparture time histogram is a measure of the cell spreading.

Finally, Figure 4.7 shows the case where the mean delay is allowed to increase

by thirty percent. In this case, almost 80% of the stream has an inter-cell spacing of $K = 0.888$, and so perhaps this stream could be called *pseudo-deterministic.*

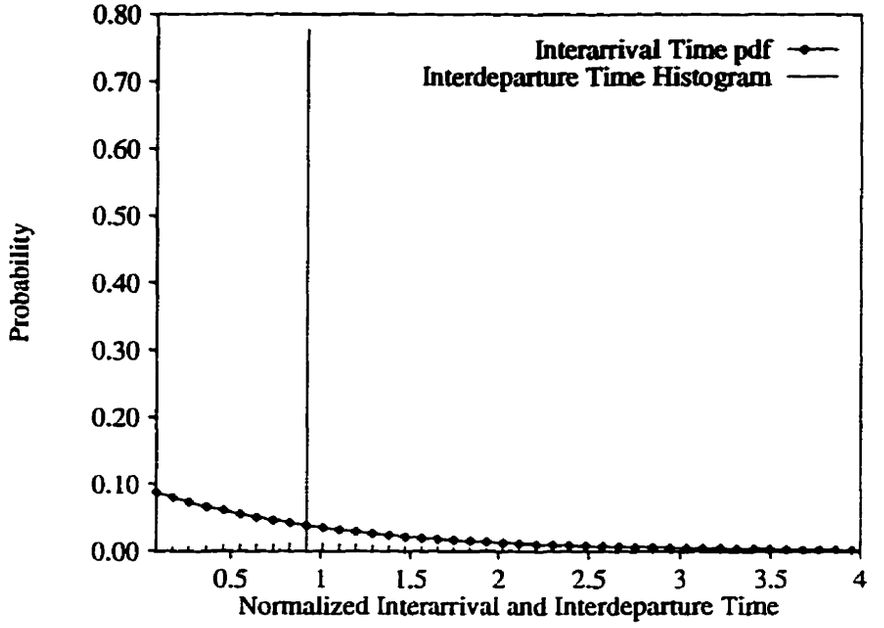


Figure 4.7: Poisson Source: $\lambda = 1.0$, $\gamma = 1.3$, $K = 0.888$

Hence, for sufficiently large γ, the shaper behaves as designed, and approximates the Ideal Shaper. Additional information is summarized in Table 4.1 for each of the three figures. Included is the shaping parameter K calculated, the squared coefficient of variation[7] of the unshaped and shaped arrival stream, CV^2 and CV_S^2 respectively, the decrease in the interdeparture time variance V_{D_i} and increase in the mean interdeparture time $\overline{D_i}$ of cells leaving the shaper and the overall increase in length of the cell stream that passes through the shaper. Also, various simu-

[7] $CV^2(x) \equiv \frac{\overline{x^2}}{\overline{x}^2}$. See [Pap84].

lation results pertaining to the shaper queue are tabulated, and as stated above, all of these entries are over ten simulation runs. The table shows that for higher

Table 4.1: Summary of Simulation Results for Poisson Source, $\lambda = 1.0$

Observation	$\gamma = 1.01$	$\gamma = 1.15$	$\gamma = 1.30$
K	0.145	0.602	0.888
CV^2	0.993	0.993	0.993
CV_S^2	0.995	0.699	0.291
Decrease in CV^2 (%)	1.940	29.540	70.700
Decrease in V_{D_i} (%)	1.941	29.535	70.696
Increase in $\overline{D_i}$ (%)	0.000	0.001	0.011
Increase in Stream Length (%)	0.000	0.001	0.006
Mean Queuing Time	0.001	0.143	2.435
Maximum Queuing Time	0.690	7.401	33.753
Mean Queue Size	0.060	0.653	3.762
Mean Maximum Queue Size	3.100	10.500	32.600
Maximum Queue Size	5.000	13.000	38.000

values of allowable mean delay increase, $\gamma\frac{1}{\lambda}$, the shaper very effectively reduces the interdeparture time variance of the cell stream at an almost negligible cost of increase in the mean interdeparture time. In fact, the length of the traffic stream hardly increases. This shows that the shaper is acting as planned, in that cells that arrive in bunches are spread into areas of the stream that have long interarrival times. Hence, for exponential interarrival times, the shaper is very effective. The cost does show up, however, in queuing time and queue size. While the cell stream, as a whole, does not increase in length, individual cells can be delayed as much as 34 time units for the case of $\gamma = 1.30$. Nonetheless, for a corresponding shaper buffer of maximum size 38 cells, this is not very large.

The reader should note that as anticipated in Equation (4.3) of Section 4.1.1.2. the two integral terms mentioned are overestimated, and thus the allowable mean delay increase specified by γ is overly pessimistic. This is caused by the assumptions made when designing the MVS.

In considering implementation issues, a look-up table of $\frac{1}{\lambda}$, γ and their corresponding value of K can be calculated off-line and accessed as the shaper operates. This has the added advantage of allowing the network provider some flexibility as experience is gained over time as the network (UNI) operates.

To complete the discussion of exponential interarrival times, refer to Figures 4.8–4.10. The first shows how the interdeparture time variance V_{D_i} decreases with

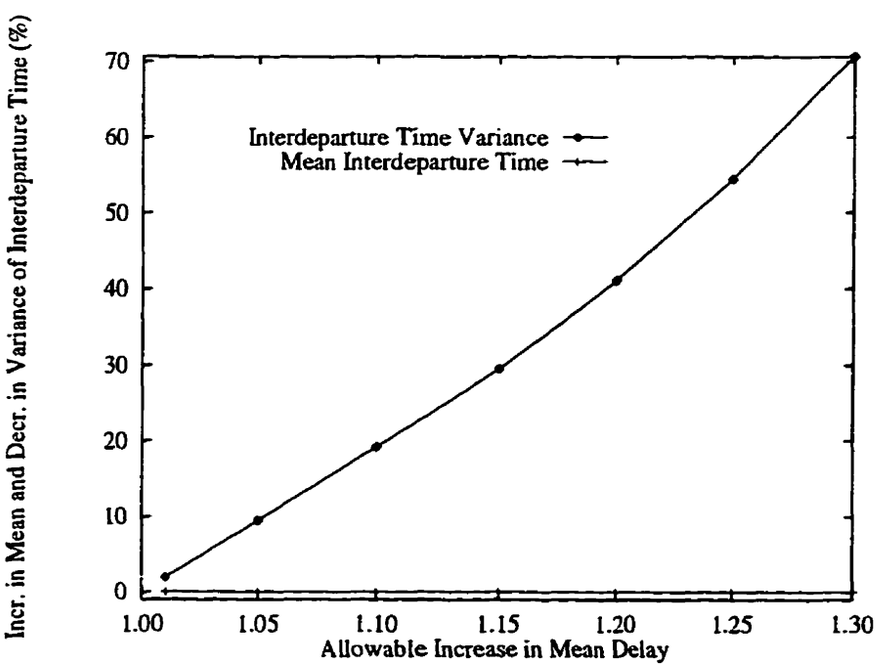


Figure 4.8: Poisson Source: $\lambda = 1.0$

increasing allowable delay, $\gamma\frac{1}{\lambda}$. The corresponding increase in mean interarrival time is also plotted, but is almost zero throughout. As can be seen, for even modest increases in allowable delay, V_{D_i} can be decreased 30–40%, which could be very beneficial in certain ATM network situations. Figure 4.9 shows the cost of this variance reduction in terms of the mean maximum queue length observed at

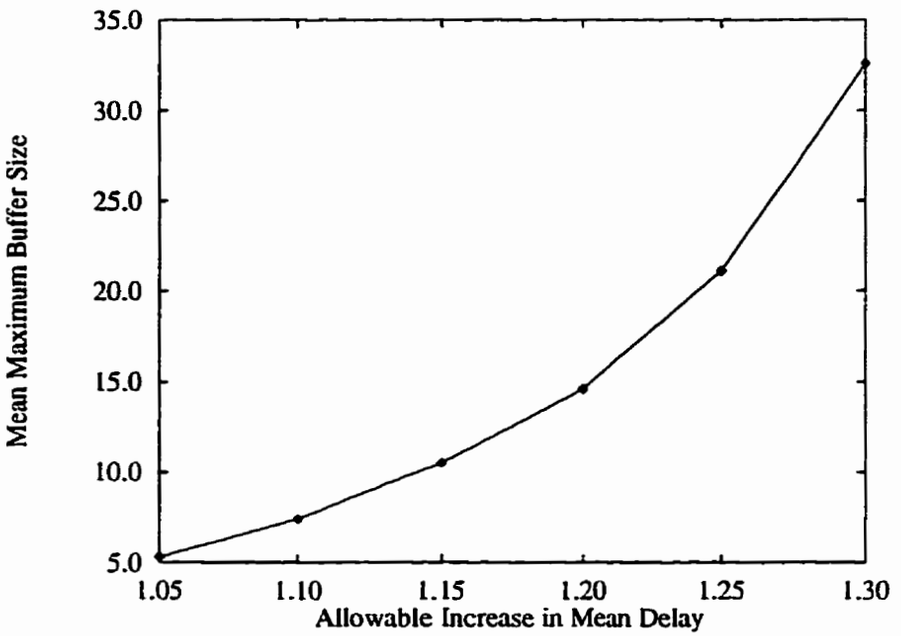


Figure 4.9: Poisson Source: $\lambda = 1.0$

the shaper over the ten simulation runs. Finally, Figure 4.10 plots the decrease in V_{D_i} versus the mean maximum buffer size, showing that it is an approximately linear function. For a unit increase in maximum buffer size, the interdeparture time variance is decreased about two and a half percent.

Moving on to the case of the on-off voice source given by Equations (4.13) and

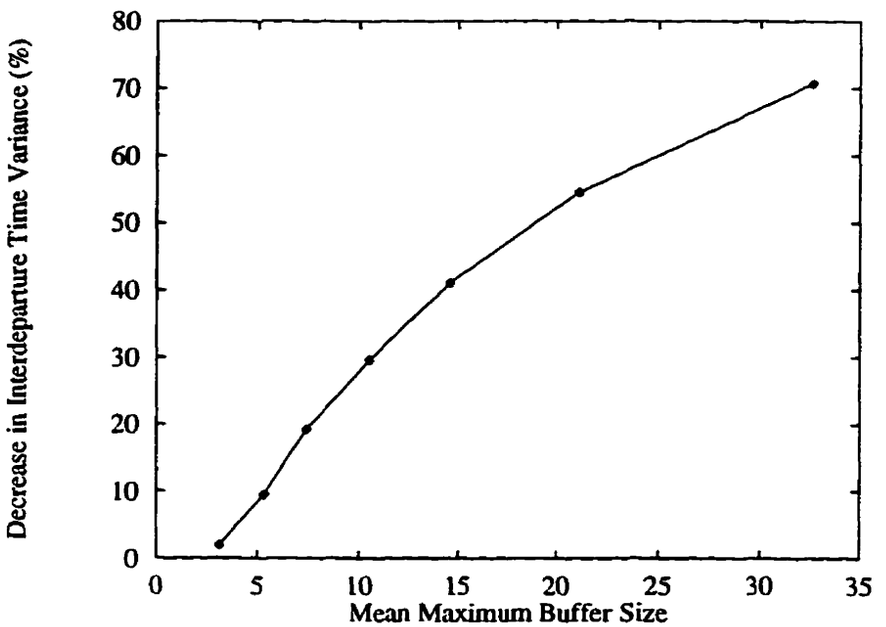


Figure 4.10: Poisson Source: $\lambda = 1.0$, $\gamma = 1.01$–1.30

(4.14), refer to Figures 4.11–4.14. The on-off source used corresponds to that in [SW86], and so the parameters used are $(\alpha\Delta)^{-1} = 22.0$, $\Delta = 16$ ms and $\beta^{-1} = 650$ ms. Figure 4.11 shows the interarrival time pdf to consist mainly of an impulse of probability at Δ, which is the output of a voice burst from the codec discussed in the reference. The silence period between the bursts is exponentially distributed, but of such a low probability that it does not show up on the graph. Nonetheless, it represents an area in the cell stream into which the cells can be spread. After $K = 0.032$ ms is calculated, the interdeparture time histogram is as shown in the figure. Only a few more percentage points are gained, so the shaped cell stream is somewhat more deterministic. While the mean queueing time and maximum

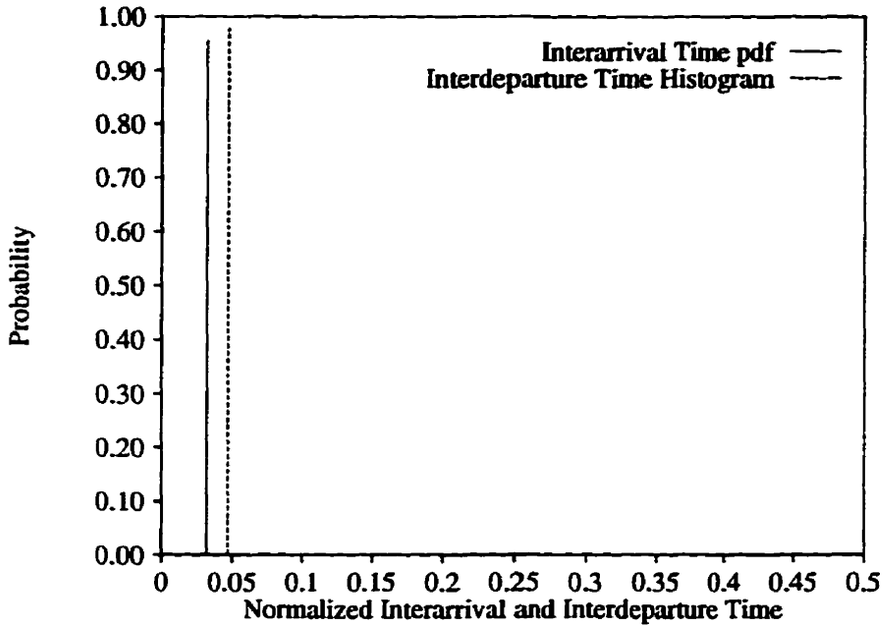


Figure 4.11: On-off Source: $(\alpha\Delta)^{-1} = 22.0$, $\Delta = 0.016$, $\beta^{-1} = 0.65$, $\gamma \approx 1.3$

queueing time are only 0.724 ms and 6.924 ms, respectively, the mean queue size is 23.347, the mean maximum queue size is 173 and the maximum queue size is 217. Thus this seemingly minor increase in the deterministic nature of the stream is coming at a high cost in queue sizes, and queueing time. In fact, the mean queueing time represents about two times the average length of the burst period, $\alpha^{-1} =$ 352 ms, and the maximum queueing time approximately twenty times. Hence, this source most likely will not react well to being shaped. Nevertheless, examining Figure 4.12 indicates that the V_{D_i} is greatly reduced even for a small increase in the allowable mean delay, whereas the $\overline{D_i}$ increases negligibly. This shows that the shaper is operating as expected. However, as Figure 4.13 shows, this comes at high

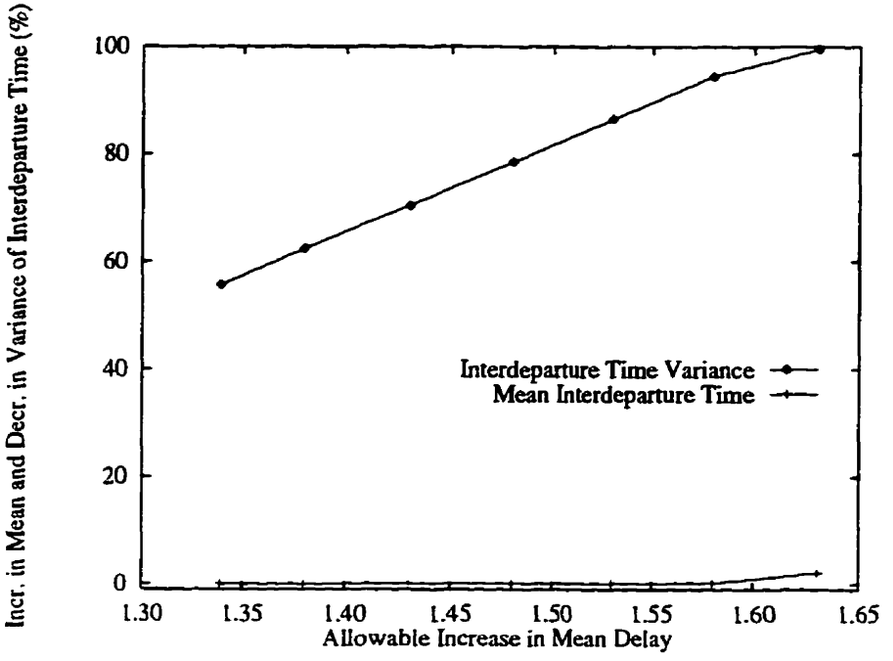


Figure 4.12: On-off Source: $(\alpha\Delta)^{-1} = 22.0$, $\Delta = 0.016$, $\beta^{-1} = 0.65$

cost in buffer size, especially for larger decreases in the variance. Lastly, Figure 4.14 indicates that the decrease in variance diminishes past $\gamma = 1.58$. This is not surprising, since the mean queuing time is 9.41 seconds, which represents many multiples of the burst and silence periods. There is no doubt that this would cause the voice decoder at the destination to fail in reproducing the analog voice signal. Hence, this particular on-off voice source is a good example of those sources which are not amenable to shaping.

This completes the presentation of results for the CT-MVS. Next, results of the DT-MVS are presented. Since the DT-MVS shaper sources are nonparametric in nature, a complete study as performed for the CT-MVS shaper and the exponential

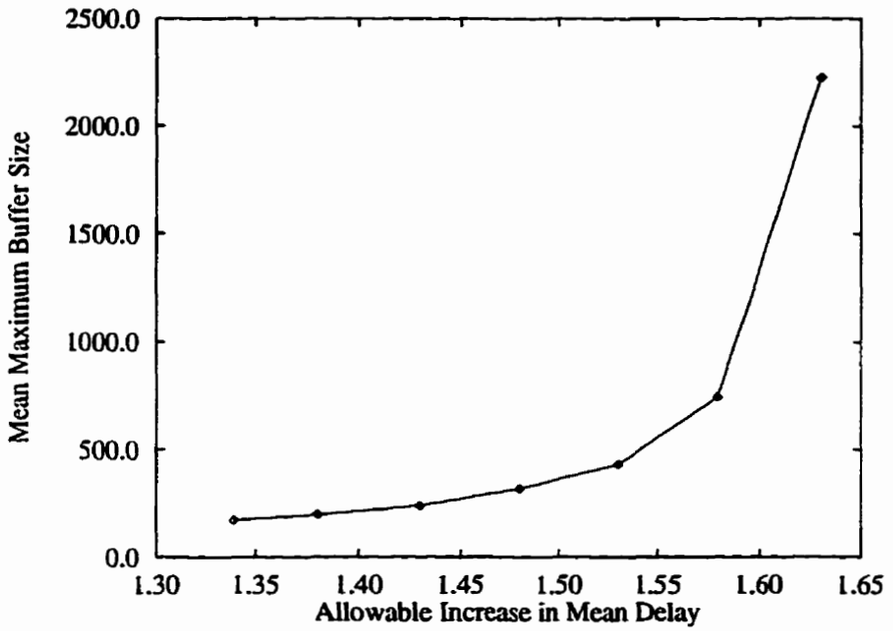


Figure 4.13: On-off Source: $(\alpha\Delta)^{-1} = 22.0$, $\Delta = 0.016$, $\beta^{-1} = 0.65$

and on-off source is not possible. Hence, only a few interesting examples are given, since in a simulation study, there are an infinite possibilities to be considered.

4.1.3.2 Discrete Time MVS

Before the main DT-MVS source examples are presented and in order to give an example of truncating and discretizing a continuous time case, shaping results are shown in Figure 4.15 which correspond to the discrete time case of Figure 4.6. As with the continuous time case, $\lambda = 1.0$, the interarrival and the interdeparture times are given by a histogram in Figure 4.15. While the interarrival time histogram line does not resemble the corresponding one in Figure 4.6, rest assured that it is a

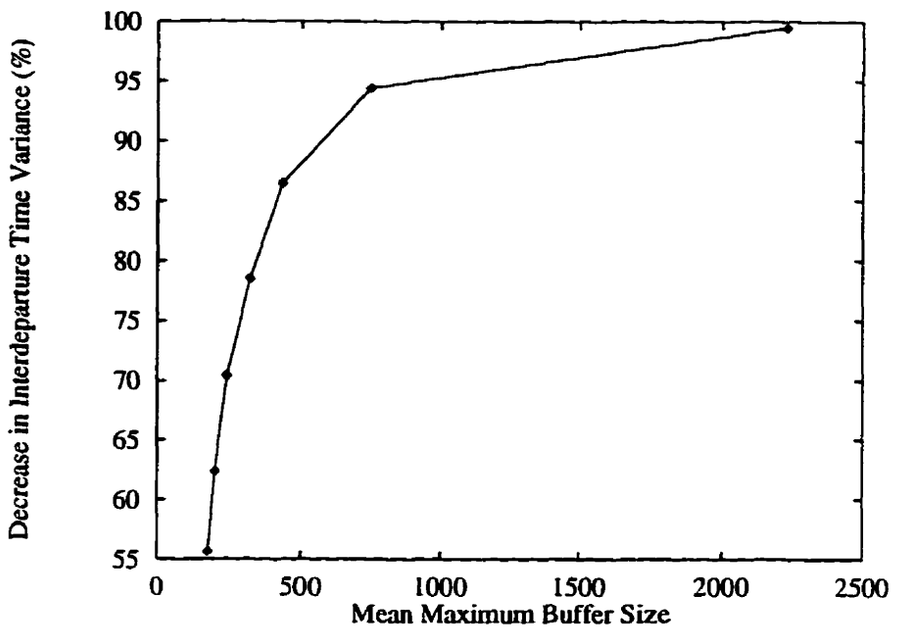


Figure 4.14: On-off Source: $(\alpha\Delta)^{-1} = 22.0$, $\Delta = 0.016$, $\beta^{-1} = 0.65$, $\gamma = 1.3$-1.63

valid distribution, that is the probability sums to one. The difference arises from the fact that in the continuous case, the pdf is plotted as a function, while in the discrete case it is plotted as a histogram, and thus the factor of the sampling rate must be taken into account. In addition, the pdf is truncated at a normalized interarrival time of approximately 9.210, which represents 99.99% of the original pdf. As expected, the results are nearly identical. In fact, the corresponding entry in Table 4.1 for $\gamma = 1.15$ is accurate to at least one decimal place. While the CT-MVS calculates $K = 0.602$, the DT-MVS calculates $K(5000, 2264) = 0.603$. Recalling the notation of Section 4.1.2, this indicates that the optimum shaping parameter value of 0.603 is obtained in section 2,264 of the discretized CDF that is

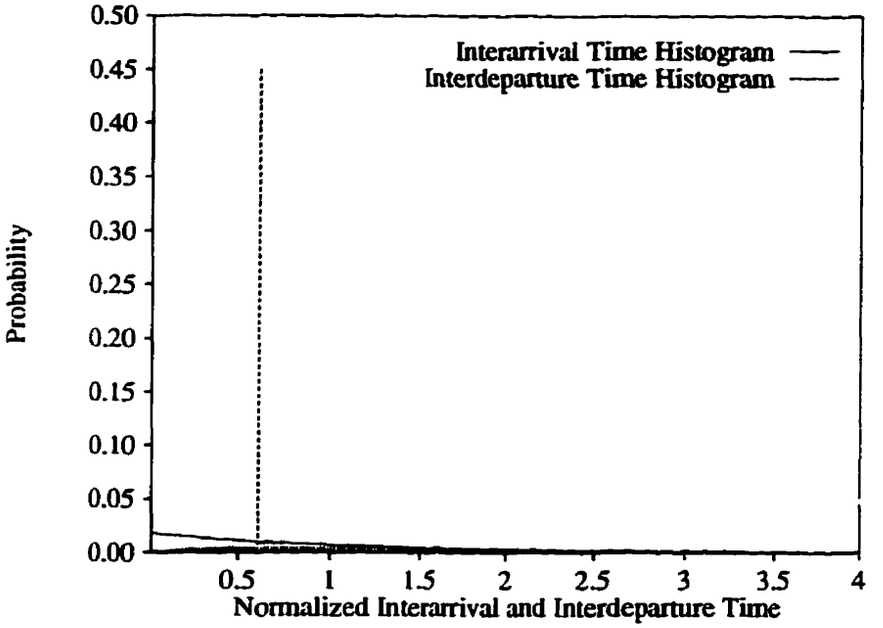


Figure 4.15: Discretized Poisson Source: $\lambda = 1.0$, $\gamma = 1.15$, $K(N,m) = 0.603$, $\xi = 0.0001$

divided into $5,000$ sections. As intuition implies, the larger the number of sections N into which the CDF is divided, the more accurate the calculation of $K(N,m)$.

To close this section, two more shaping examples are briefly presented. The first is an MMBP source, with two states. The mean arrival rate of cells in states one and two are $\lambda_1 = 0.01$ and $\lambda_2 = 20.8$, respectfully. The normalized holding times are $H_1 = 1.0$ and $H_2 = 0.05$. This gives an overall mean arrival rate of $\lambda = 1.0$. The interarrival and interdeparture time histograms are shown in Figure 4.16. The optimum value of the shaping parameter is calculated to be $(1000, 22) = 0.603$. Since MMBP sources are similar in nature to poisson sources, it is not surprising to see the DT-MVS shaper perform as well as in the case of the poisson source of

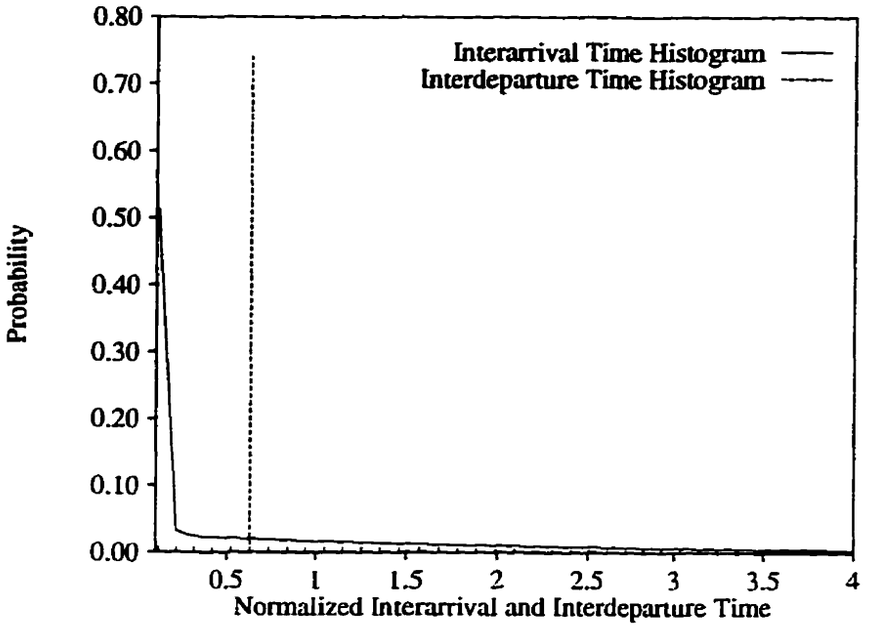


Figure 4.16: MMBP Source: $\lambda_1 = 0.01$, $\lambda_2 = 20.8$, $H_1 = 1.0$, $H_2 = 0.05$. $\gamma = 1.3$, $K(N, m) = 0.573$

Figure 4.7, which also has $\lambda = 1.0$ and $\gamma = 1.3$. However, since the MMBP source is burstier, the shaping algorithm takes this into account by observing the interarrival time histogram, and reduces the optimum value from $K = 0.888$ as calculated in the continuous time case. Summarizing the simulation results, the interdeparture time variance is reduced 46.11% with an increase in the mean interdeparture time of 0.003%. This comes at a cost of mean and maximum queuing times of 1.143 and 23.455, with mean queue size, mean maximum queue size and maximum queue sizes of 2.806, 30.8 and 41, respectively. Again, note that over 70% of the cell arrivals now leave the shaper with a deterministic cell spacing. Also, the increase in stream length is a negligible 0.001%. Hence, the MMBP source with widely different arrival

rates and holding times of its two states is shaped to a near deterministic source.

The final results presented are those pertaining to a packet train source, in particular one with geometrically distributed on and off periods. Using the notation of Section 1.6.6.4, define $\mathrm{PT}_G(x,y) = \Psi_{OO}(\mathcal{G}(x),\emptyset,\mathcal{G}(y))$. In particular. the interarrival time and interdeparture time distributions of a $\mathrm{PT}_G(2,2)$ cell stream is shown in Figure 4.17. For this source, the maximum interarrival time is seventeen

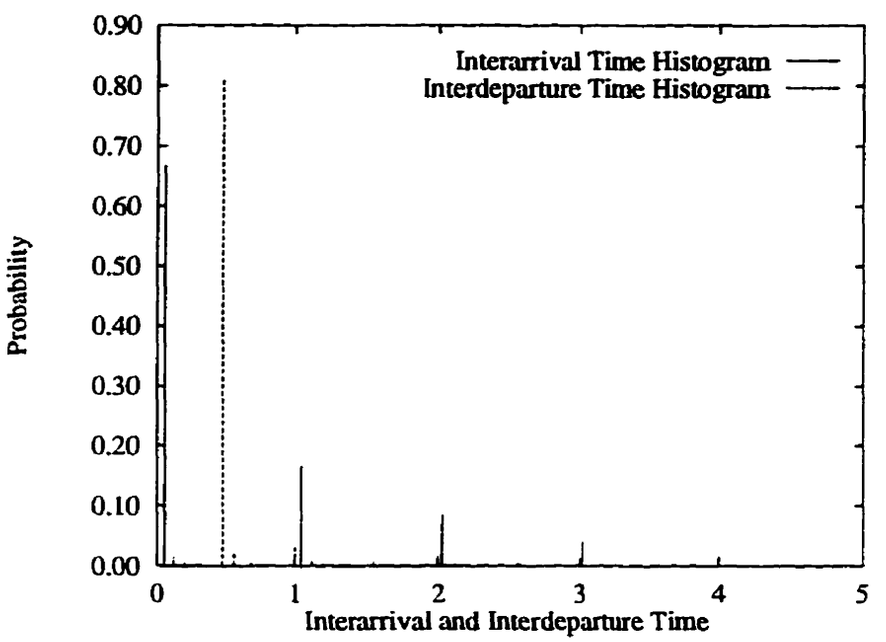


Figure 4.17: $\mathrm{PT}_G(2,2)$ Source: Mean burst size 2, mean silence size 2, $\gamma = 1.3$, $K(N,m) = 0.451$

cells, which implies eighteen sections in the optimization of $K(N,m)$. As a result, $(18,2) = 0.451$, which as can be seen increases the amount of deterministic cell spacing from just over 65% to about 80%, corresponding to a 62.62% decrease in the interdeparture time variance with just a 0.5% increase in the mean. The cost

is a mean and maximum queueing times of 1.685 and 15.104, and a mean, mean maximum and maximum buffer size of 4.589, 23.5 and 34, respectively. Thus, since this source is burstier still than the MMBP the interdeparture time variance is reduced more, with similar results at the shaper buffer.

Summarizing the results, then, of the Minimized Variance shaper, it can be said that it is an effective tool in attempting to create a deterministic cell stream from a probabilistic one. In particular, poisson, MMBP and PT sources are likely candidates for this shaping scheme. In all three cases the original source is shaped into a nearly deterministic source with interarrival times K or $K(N,m)$. Of course. the shaping has come at a price of the size of the buffer of the shaper, and the slight increase in interarrival times. Both these can be increased or reduced by varying γ. On the other hand, the voice source did not fair well when shaped, and thus this method should not be employed in these cases. One interesting side-bar that arises from the shaping of the on-off voice source is that shaping a near-deterministic stream results in another near-deterministic stream. This intuitive statement implies that after a point, further shaping results in diminishing returns, especially with respect to the increased costs of queueing delays and buffer sizes at the shaper.

4.1.3.3 Comparison with Other Shapers

For completeness, an attempt is made to compare the results of the MVS to other methods found in the literature. However the comparisons made are necessarily relative, since as previously stated, the goal of the MVS is to reduce the inter-cell variance in a traffic stream to zero, whereas often the goal of shaping in the literature

is to limit Cell Delay Variation (CDV). Other difficulties in comparison arise due to the fact that no standard traffic stream is used as a point of comparison, and so the recreation of the exact cell streams other methods employ is usually impossible.

Considering the results given in [Bro92], it can be seen that a direct comparison is impossible due to the fact that the results are normalized by the shaping parameter used. This is reasonable in the context of this work, since no optimization on the shaping parameter is performed. In addition, the traffic stream shaped is an aggregrate of on-off voice sources and data sources with cell arrivals obeying a Poisson distribution. Unfortunately, a cell interarrival time histogram is not given. Hence, the results of the MVS with respect to the shaping of a Poisson source are most likely similar. Therefore, refer to the results given in Table 4.1. Considering the best results of [Bro92], CV^2 is reduced by approximately 85% at a cost of increasing the stream length by about 7%. This seems to arise from the fact that there is an increase in mean interdeparture time of about 5%, and the mean queueing time is just under 1 ms, which represents 365 cells at $C_L = 155$ Mbps. Therefore, it appears that the MVS compares quite favourably to this shaping method, since CV^2 is reduced only 15% less at no cost with respect to the increase in stream length.

Comparison to the results of [BGSC92] is somewhat easier, since interarrival and interdeparture time histograms are given (Figures 9 and 10). In this case seven CBR traffic streams are perturbed by other arrival processes at a series of five nodes, and then the aggregate of these processes is shaped at a sixth common node. Since the perturbation is Bernoulli which results in an interarrival time histogram which has

the characteristics of a Normal distribution, the Poisson source results of the MVS are somewhat comparable. Because a Normal distribution can be considered to be a "two sided" exponential distribution, there is a larger proportion of long interarrival times in the case of the MVS results of, for example, Figure 4.7. Since a reduction of long interarrival times and an increase in short interarrival times implies a move from hard shaping to soft shaping, the MVS results for a Poisson source can be considered to be a lower bound to the results for a Normal source. Figure 10 of [BGSC92] shows that this shaping method is able to produce a deterministic source just over 70% of the time, as compared to almost 80% of Figure 4.7 for the MVS. Thus the MVS again compares very favourably, especially considering, as discussed above, that the MVS results are even better when shaping the type of source used in the citation. Unfortuantely, no other performance results are given. Therefore, it can be stated with some confidence that the results of the MVS compare quite favourably with some of the more established methods of the literature.

The following section presents the algorithm of a shaper which has the same goal as the MVS, namely to create as deterministic a cell stream as possible. However, its method for achieving this goal is quite dissimilar from that discussed in the preceding sections. In addition, the Burst-oriented shaper has the ability to unshape cell streams.

4.2 The Burst-Oriented Shaper

There are two main goals of the Burst-Oriented Shaper (BOS). The first is to take an arbitrary on-off cell stream and shape it to as near a deterministic cell stream as possible. If traffic streams presented to the network are near deterministic, the provisioning of downstream network devices is simplified, as mentioned in Section 1.3.2, since the inputs to these devices are well defined. As mentioned, this should ease network management, since CAC, UPC and cell scheduling at ATM switches is also simplified.

The second goal of the BOS is to provide the user with a network connection which is as transparent as possible. Thus, the BOS presents the destination UNI's AAL with, ideally, the same bursty on-off traffic stream created by the segmentation at the AAL of the source UNI. In other words, the BOS algorithm is designed to both shape and unshape the stream. By doing so, the problems encountered due to Cell Delay Variation (CDV) are reduced. That is, since the destination AAL presents higher network layers the same bursty stream that is shaped, any delays caused by contention at ATM switches or shaping itself may be alleviated.

It should be noted that since the BOS is designed to operate after cellization of the traffic stream at the AAL, as described in Section 1.6.4, it will most surely operate only on on-off or packet train cell streams. This idea is basic to the operation of the BOS, since it is based on a simple premise: if every burst can be uniformly spread into the silence which follows it, then the traffic stream will be completely deterministic, and so the BOS will behave like the Ideal Shaper.

4.2.1 The Shaper Model

In order to achieve the first goal mentioned in the previous section, the BOS attempts to maintain the shaped traffic stream at a target rate, R_T cells per unit time, for as long as possible. This is accomplished by "spreading out" a burst into the silence period which immediately follows it. Since the traffic stream to the shaper is unknown, in order to determine where a burst starts and ends, a reference point is required. This reference is implemented in the form of a "window" superimposed on the traffic stream, as shown in Figure 4.18. Note that this is completely analogous to the traffic window used in traffic classification, introduced in Section 2.1.4.

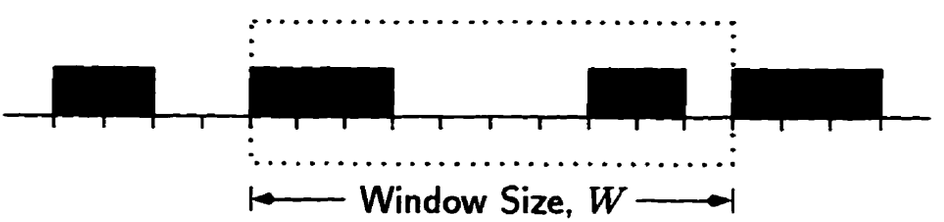


Figure 4.18: The Burst-oriented Shaper Window

As mentioned at the beginning of this chapter, the experience gained with traffic classification is directly applied to the BOS. In fact, since the idea of spreading a burst into the following silence is based on an implied PT traffic primitive, one could say that the BOS has traffic classification "built in." This is the reason the BOS does not require information about the type of traffic class it is operating on: there is only one class, namely on-off. The window is "parsed" to locate the first burst and the silence which immediately follows it. Of course, in order for the window to be parsed, the cells contained in the window must have already arrived

to the shaper; these cells are stored in the shaper's queue.

To achieve the second goal, the BOS embeds unshaping information into the shaped traffic stream. This takes the form of an unshaping parameter, Λ. Thus, the transmission of the unshaping parameter is overhead to the data transmission inside the network. Figure 4.19 zooms in on the shaping window of Figure 4.18 in order

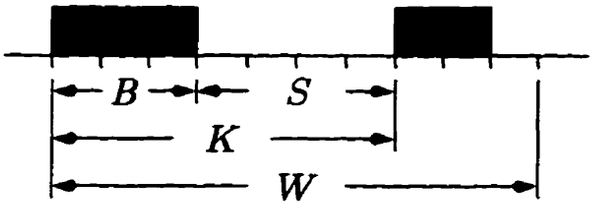


Figure 4.19: Burst-oriented Shaper Parameters

to depict the parameters that are employed by the BOS shaping and unshaping algorithms. The parameters are defined as follows.

B: The size of the *first* burst of the window, where the burst is defined to start at the left-hand-side (LHS) of the window.

S: The size of the *first* silence period of the window, where the silence period is defined to start immediately after the first burst, B.

K: The shaping period, usually $K = B + S \leq W$.

S^*: The size of the *first* silence period of the window, when the silence period *starts* at the LHS of the window; that is, before the *first* burst. Note that $S^* \leq W$.

W: The size of the shaping window.

Λ: The unshaping parameter. For shaping period i, $\Lambda_i = (B_i, S_i)$ is a doublet representing the size of the burst and silence period of the unshaped (original) traffic stream. This is utilized by the unshaping algorithm to restore the traffic stream.

R_T: The target rate at which to shape the traffic stream, in cells per unit time.

Q: The number of cells queued at the shaper.

If the payload field of an ATM cell is used to represent Λ, and assuming eight bytes are reserved for internal use by the BOS, then evenly dividing the remaining 40 bytes between the representation of B and S will allow a maximum window size of $W = 2^{20\times 8} = 1,048,576$, which is greater than any window sized envisioned. If $C_L = 150$ Mbps, this represents a maximum burst length of about 2.68 seconds, which is a very long time — even for a low rate voice source. Additionally, one of these unshaping parameter cells is required approximately every W cells, since in general $K \leq W$. Thus, the larger the shaping window size, the lower the overhead of the algorithm, which can be approximated by $\frac{1}{W+1}$.

4.2.2 The Shaping Algorithm

Algorithm 4.2.1 states the Burst-oriented shaping method. Initially, the shaper stores cells in its queue as it fills the shaping window, W, as shown in Figure 4.19. As bursts and silence periods are parsed and the traffic stream shaped, and as new cells arrive to the shaper as described in Step 2 of Algorithm 4.2.1, the shaping window is updated to reflect these changes. In other words, the shaper keeps a

"sliding window snapshot" of the traffic stream.

Algorithm 4.2.1 The BOS Shaping Algorithm.

1. *Shaper setup.*
   (a) *Determine the rate at which the traffic stream is to be shaped,* $\mathsf{R_T}$, *from UPC.*
   (b) *Delay the traffic stream a minimum of* K *and a maximum of* W *cells, storing the cells in the shaper queue,* Q.
   (c) *Initialize the shaping period counter,* $i = 1$.
   (d) *Parse the shaper window to determine the shaper state, and thus the unshaping parameter for the first shaping period,* $\Lambda_1 = (B_1, S_1)$.
2. *Shaper operation.*
   (a) *Output the unshaping parameter for this shaping period,* $\Lambda_i = (B_i, S_i)$.
   (b) *Output queued cells at rate* $\mathsf{R_T}$ *for* K *cell times, while* $Q > 0$. *When* $Q = 0$, *generate slots until time* K *cell times have elapsed. Store any newly arrived cells in the shaper queue.*
   (c) *Shift the shaper window along the traffic stream by* K *cells.*
   (d) *Increment the shaping period,* $i = i + 1$.
   (e) *Parse the shaper window to determine the shaper state, and thus the unshaping parameter,* $\Lambda_i = (B_i, S_i)$.
3. *Repeat step 2. until the traffic stream ends.*
4. *Output cells at rate* $\mathsf{R_T}$ *until* $Q = 0$, *if necessary.*
5. *Transmit the end-of-shaping marker.*

The action of the shaper is to wait until a burst starts at the LHS of the shaping window, and then compute B, S, and K. It then outputs the unshaping parameter Λ *before* outputting the shaped burst. In order to better describe the operation of the basic shaper, there are five states in which the shaper can be, enumerated below

and depicted in State Diagram 4.2.1. Assume $W = 10$ in the following, and the usual "1" and "0" notation introduced in Section 1.6.4. In each state, the shaper outputs cells at the target shaping rate R_T until $Q = 0$, at which point the shaping "breaks." That is, since no new cells have arrived and the shaper has exhausted its supply of cells stored in the queue, the BOS has no choice but to allow a long interdeparture time result.

State 1: Burst B and silence S defined. This is the "normal" or planned state of operation of the shaper. The shaper outputs at rate R_T for $K = B+S < W$ cell times. Q may increase or decrease. Example window: 1111000110. The unshaping parameter is $\Lambda = (B, S)$.

State 2: B defined, S undefined. In this case the silence period is *set* to $S = W-B$, and $K = W$. Q may increase or decrease. Example window: 1111000000. $\Lambda = (B, W - B)$.

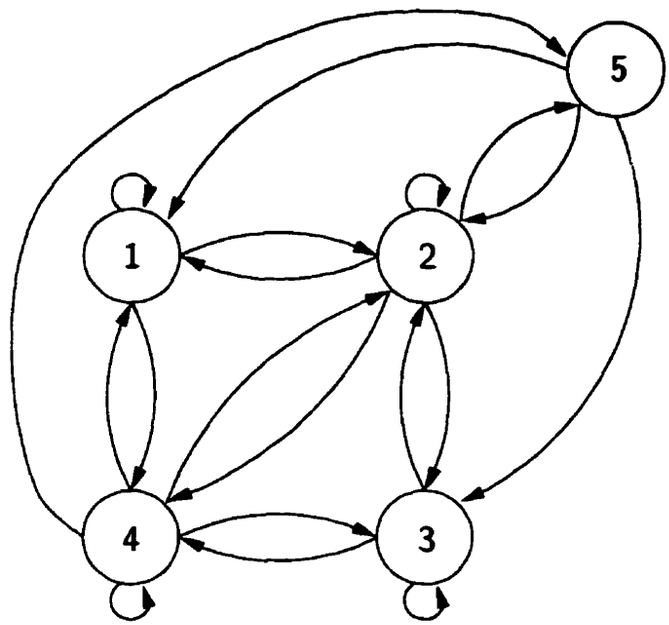
State 3: B undefined, S defined. Here $B = W$ and $S = 0$. In order to ensure that the next shaping period starts with a burst, $K = W - 1$ (instead of $K = W$, as one might expect from the preceding two states). Q will increase. Example window: 1111111111. $\Lambda = (W - 1, 0)$.

State 4: B undefined, S undefined. In this case $B = 0$ and $S = W$, and so $K = W$. Q will decrease. Example window: 0000000000. $\Lambda = (0, W)$.

State 5: Due to the actions of States 2 and 4, the window may not start with a burst, as required, and so this state simply corrects the situation. K is

set to S^*, the time required for the next burst to start at the LHS of the window. Q will decrease. Example window: 0001111000. $\Lambda = (0, S^*)$.

State Diagram 4.2.1 State Diagram for the BOS



Once the traffic stream comes to an end, the shaper maintains the transmission of cells stored in its queue at the target rate R_T, until the queue is empty. Then it transmit an "end of shaping" marker in order to inform the destination unshaper that the stream has ended.

4.2.3 The Unshaping Algorithm

The operation of the unshaper is quite simple, as Algorithm 4.2.2 shows. Once the unshaping parameter Λ_i is received, the unshaper waits until B_i cells are queued and

then presents them to the AAL, followed by S_i slots. If cells start to arrive before the unshaping parameter, then the unshaping parameter has been lost.[8] In this case, the best course of action is for the unshaper to presents cells to the destination AAL as they are received. While a multitude of methods could be devised, such as repeating the last Λ until the a new one is received, the reasoning here is that if the correct source information is lost, it is better not to make assumptions about the source characteristics. Also, note that in W slot times, another unshaping parameter should arrive, and so this situation will not continue indefinitely. Hence, the unshaping algorithm is quite robust in the face of lost unshaping parameter cells, since each Λ applies to at most only W cells.

Algorithm 4.2.2 The BOS Unshaping Algorithm.

1. *Unshaper setup.*
   (a) *Initialize the unshaping period counter,* $i = 1$.
2. *Unshaper operation.*
   (a) *Receive the unshaping parameter for this unshaping period,* $\Lambda_i = (B_i, S_i)$. *If cells arrive instead of* Λ, *then the* Λ *has been lost. Present the cell stream "as is" to the AAL, until a* Λ *arrives. Call this* Λ_i.
   (b) *Wait until* B_i *cells have been queued.*
   (c) *Present a burst of size* B_i *cells followed by* S_i *slots to the AAL.*
   (d) *Increment the shaping period,* $i = i + 1$.
3. *Repeat step 2. until the end of shaping marker has been received.*

The unshaping period i allows the use of sequence numbers for Λ, but this is not

[8]The cells cannot arrive out of order, since ATM offers connection-oriented service.

necessary. As can be seen from Algorithm 4.2.2, after the loss of a Λ the method is self-correcting. In addition, it is assumed that the transmission media in an ATM network provides virtually error free transmission, as is the usual practice in the literature, and so the only way that the unshaping parameters can be lost is due to congestion at network nodes. In order to avoid this occurrence, the eight bytes of the unshaping parameter cell reserved, as stated in Section 4.2.1, could be used to identify the Λ cell as a high priority traffic management cell. The following section discusses how the shaping window size W can be determined, and proposes an algorithm based on traffic characteristics at the AAL to do so.

4.2.4 Determination of Window Size, W

As stated in Section 4.2.1, since the BOS is developed from the insights gained in traffic classification, many of the factors determining the size of the traffic window of the primitive classifier hold. Thus, recall the discussions of Sections 2.1.1 and 2.1.4, which state, in the case of the BOS, that the shaping window size represents a trade-off between the ability to shape and the delay introduced into the traffic stream. One major difference, however, is the consideration of training time. Since the BOS does not use neural networks, W can be much larger than is the case with the primitive classifier. Ultimately, an upper bound on the length of the shaping window is the desire to store the unshaping parameter $\Lambda(B, S)$ within one cell, as mentioned at the close of Section 4.2.1. Perhaps a more telling bound is the amount of computational power required to parse the shaping window. While it is not in the scope of this thesis to make accurate judgements about the feasibility

of implementing the BOS on a Very Large Scale Integration (VLSI) chip, it does seem reasonable that very large shaping windows can be supported. Since the BOS is divided into five states, separate modules could be implemented for each state, with a sixth module for controlling logic, which determines the current state. For example, to determine the length of the burst which starts at the LHS of the shaping window, the module can be implemented with a cascade of AND gates. In fact, all the parameters of the BOS can be determined in this way. Therefore. it may well be that implementation issues do not limit the size of W. Thus, another limit is required.

One such limit results from the fact that as long as W contains a complete burst and silence period, as a consequence of the shaping algorithm, it need be no larger. Thus it is possible, given knowledge of the source, to limit W in a statistical sense, as follows. Given a mean burst length and mean silence length, this information can be converted into a mean number of ATM cells in a burst and slots in a silence period, after cellization at the AAL. With this information, the shaping window can be dimensioned so that a property such as the following holds true,

$$\Pr\left\{\overline{\text{burstlength}} + \overline{\text{silencelength}} > W\right\} < 0.01. \tag{4.38}$$

In other words, this equation states the desire of the network provider to capture 99% of all burst and silence periods within the shaping window W.

To evaluate Equation (4.38) requires intimate experience with the AAL, since burst and silence length distributions or at least histograms are required. Nonetheless, keeping these values variable, a method which can estimate the upper bound

is given next, based on a burst and silence length histogram being available. Now,

$$
\begin{aligned}
&\Pr\left\{\overline{\text{burst length}} + \overline{\text{silence length}} > W\right\} \\
&= 1 - \Pr\left\{\overline{\text{burst length}} + \overline{\text{silence length}} \leq W\right\} \\
&= 1 - (W-1) \cdot \sum_{i=1}^{W-2} Pr\left\{\text{burst length} = i\right\} \cdot Pr\left\{\text{silence length} = W - i - 1\right\} \\
&+ W\Bigg(\sum_{j=1}^{W} Pr\left\{\text{silence length} = j\right\} \cdot Pr\left\{\text{burst length} = W - j\right\} \\
&+ \sum_{k=1}^{W-2} \sum_{i=1}^{W-k-1} \Big[Pr\left\{\text{silence length} = k\right\} \cdot Pr\left\{\text{burst length} = i\right\} \\
&\qquad \cdot Pr\left\{\text{silence length} = W - k - i\right\} \Big] \Bigg),
\end{aligned}
\tag{4.39}
$$

which accounts for all three cases of a burst starting a window followed by a silence period, a silence period starting the window followed by a burst, and a silence period starting the window followed by a burst and yet another silence period. After some thought, the reader should realize that these three situations are a result of the states of State Diagram 4.2.1. Completing the method, begin with $W = 3$, or some other suitably small number, based on the expected size of the shaping window. Iterate Equation (4.39) until it evaluates to less than 0.01, incrementing W after each iteration.

The following section discusses some of the results of shaping various sources using the BOS.

4.2.5 Results

This chapter concludes with a discussion of the results of the BOS. A software suite written in C++ is used to implement the BOS shaping algorithm. As in Section 4.1.3, confidence intervals are omitted for clarity. For each simulation run, there are 1,000,000 cell arrivals. The BOS unshaping algorithm is not implemented, for two reasons. Firstly, if the network does not drop any Λ, then the algorithm will operate as designed. Thus, the shaped and unshaped cell streams are the same. One way of measuring this could be by using the cross-correlation function.[9] Secondly, if the network delays or drops Λ, or delays the cells of the shaped stream itself, then obviously the unshaped cell stream will differ from the original. However, this is an external effect on shaping, which all shapers must contend with, and thus is not germane to this discussion.

Since the BOS is designed to shape on-off traffic streams, those with on and off distributions which are geometrically distributed are presented. The general on-off sources chosen are of the type $\mathrm{PT}_G(x,y) = \Psi_{OO}(\mathcal{G}(x), \emptyset, \mathcal{G}(y))$, using the notation of Section 1.6.6.4. The results appear in Figures 4.20–4.23. For these results, the shaping window size is $W = 10$ and the target shaping rate is $\mathsf{R}_\mathsf{T} = 0.5$, that is a cell followed by a space, or "101010101," using the notation of Section 1.6.4. In addition, for the four figures mentioned, the mean on burst length and mean off burst length, or mean silence length, of the on-off sources is increased equally, so that the mean arrival rate is kept constant. In particular, the $\mathrm{PT}_G(5,5)$, $\mathrm{PT}_G(10,10)$, $\mathrm{PT}_G(15,15)$, $\mathrm{PT}_G(20,20)$, $\mathrm{PT}_G(25,25)$ and $\mathrm{PT}_G(30,30)$ on-off sources are stud-

[9]The cross-correlation function, $C_{XY}(x,y)$, measures the "sameness" of two stochastic processes, X and Y. Refer to [Hay88] for a definition.

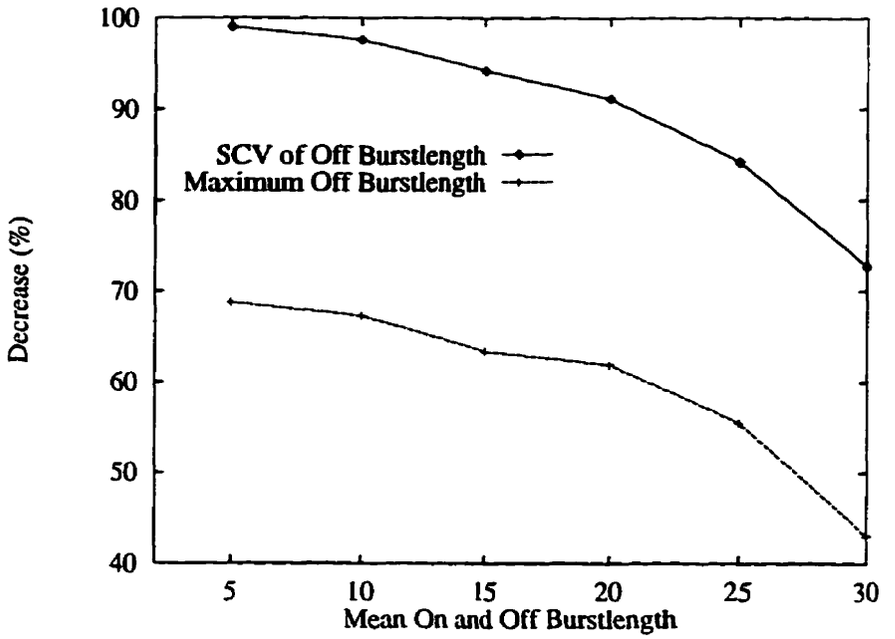


Figure 4.20: Effect of Increasing Unshaped Burst and Silence Lengths on CV^2 and Maximum Shaped Silence Length, $W = 10$, $R_T = 0.5$

ied. Figure 4.20 shows the effect on the squared coefficient of variation,[10] CV^2, when the mean on burst length and mean off silence period of the sources to be shaped are increased. Note that when the mean is low relative to the shaping window size, as with the $PT_G(5,5)$, almost ideal shaping occurs. The value of CV^2 is reduced to almost zero. Also shown in the figure, the mean off burst length is decreased by 70%. Since the goal of the BOS is to spread bursts into silence periods, the decrease in the mean silence length that results after shaping is a good measure of performance. Figure 4.21 shows the effect of the BOS on the burst length

[10]See Section 4.1.3 for the definition of CV^2. Note that CV^2 is denoted as SCV in Figure 4.20.

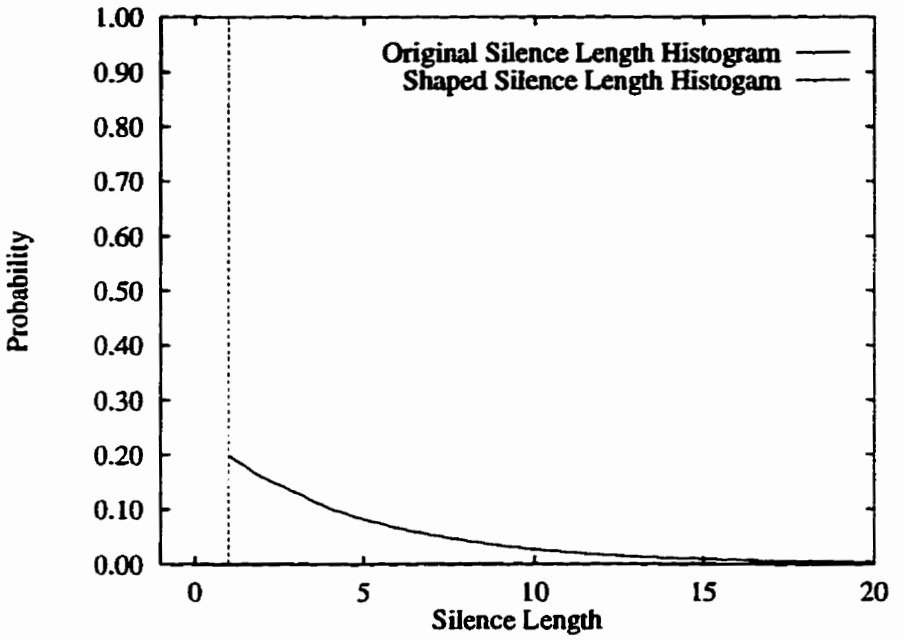


Figure 4.21: $PT_G(5,5)$ Source: Original and Shaped Silence Length Distributions, $W = 10$, $R_T = 0.5$

distribution of the original[11] and shaped cell streams. As can be seen from the figure, the results indicate that 99.98% of the silence periods are shaped to a value of one, which indicates that the target shaping rate is satisfied. Thus, the shaped cell stream is almost one hundred percent deterministic, with intercell spacing of one slot. Note that this result holds true for the other sources as well; in no cases do the shaped silence length distributions have less than 99.9% of the probability. This, however, must come at some cost. Figure 4.22 shows the effect of increasing the mean on burst length and mean silence length on the overall cell stream length.

[11]In this context, original is used instead of the usual unshaped, since unshaping has a particular meaning with respect to the BOS.

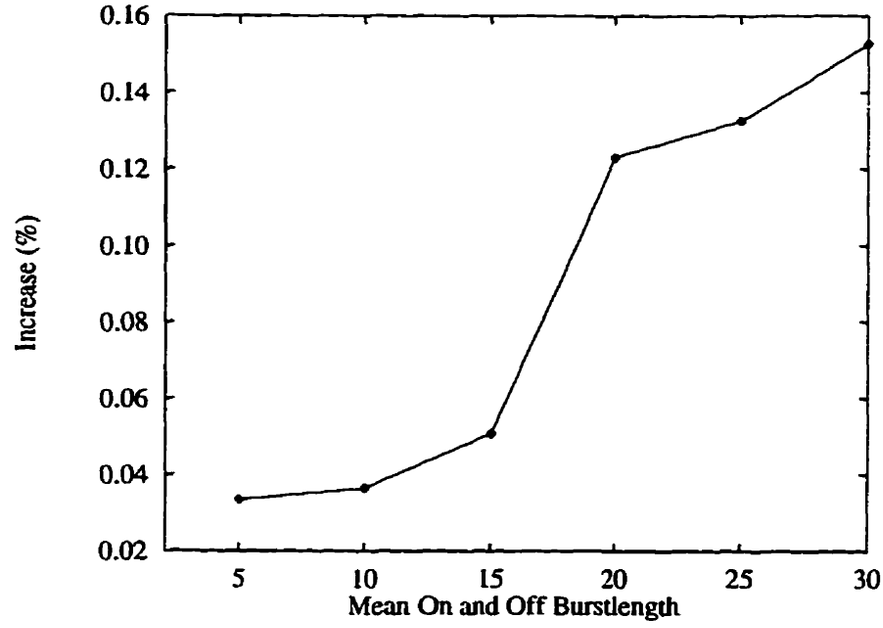


Figure 4.22: Effect of Increasing Unshaped Burst and Silence Lengths on Cell Stream Length, $W = 10$, $R_T = 0.5$

As can be seen, the cell stream length is virtually unaffected by shaping, increasing a modest 0.15% at most. Thus, the cost of shaping must appear elsewhere in the BOS system. Finally, Figure 4.23 shows that fairly heavy queuing occurs at the BOS, in terms of the maximum queue length, Q. The higher values may be prohibitive for some applications.

Referring again to Figure 4.20, it is interesting to see the diminishing returns in terms of CV^2 and maximum silence length, as the mean burst and silence lengths of the original cell streams increase. This is counterintuitive, because as stated above, the silence length distributions of the sources have almost all of their probability mass at one. Nonetheless, the discussion of the sizing of W in Section 4.2.4

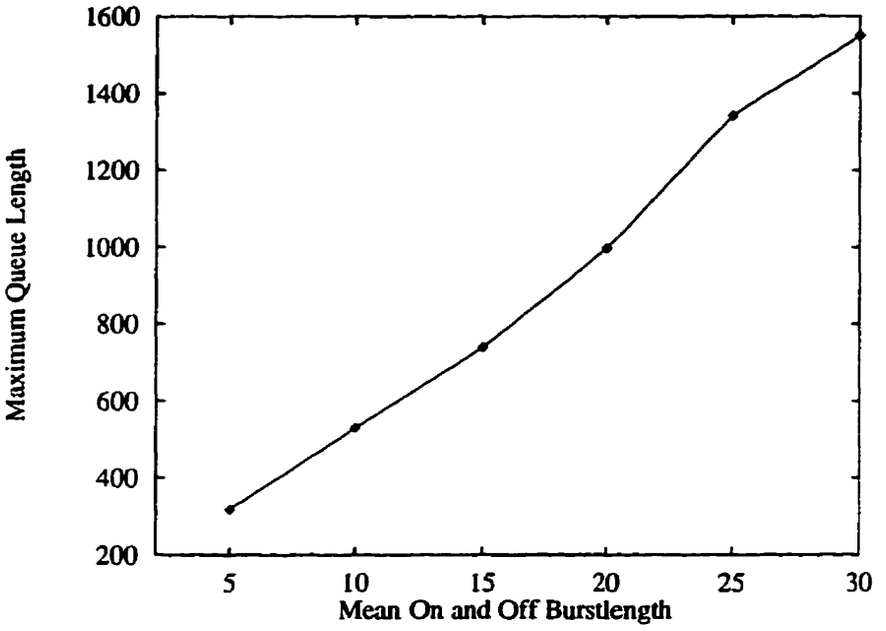


Figure 4.23: Effect of Increasing Unshaped Burst and Silence Lengths on Maximum Shaper Queue Length, $W = 10$, $R_T = 0.5$

anticipates this result. The explanation is as follows, comparing the $PT_G(5,5)$ and $PT_G(30,30)$ source results. As the mean burst and silence lengths of cell stream entering the shaper increase with respect to the shaping window size W, the shaper tends not to operate in State 1, as described in Section 4.2.2, but rather in States 3, 4 and 5. This implies that the shaper is no longer operating as intended. During the long bursts, the queue increases quickly, and during long silences, the queue empties quickly — this is the heart of the problem. Notwithstanding the fact that the queue is larger in the case of the $PT_G(30,30)$ source than in the case of the $PT_G(5,5)$ source, since the mean silence length is six times larger for $PT_G(30,30)$ source, there is a much higher chance of the queue emptying during a very long

silence period; the mean maximum silence length of the $\mathrm{PT}_G(30,30)$ source is 179, or almost eighteen times W, whereas for the $\mathrm{PT}_G(5,5)$ source, it is only 20, or two times W. And herein lies the reason for the increase in shaped silence length variance as the original silence length increases relative to W. In other words, the variance in shaped silence length *increases*, even though the maximum queue size is very large. The results bear this out. For the $\mathrm{PT}_G(5,5)$ source, the mean shaped silence length is 1.00097 as expected, and the variance is low, at 0.00776. However, for the $\mathrm{PT}_G(30,30)$ source, the mean is also 1.00417 as expected but the variance has increased to 0.271835, a thirty-five fold increase. The conclusion is this: for best operation of the BOS, the shaper window size W must be at least on the same order as the original mean burst length plus the original mean silence length, as Equation (4.38) implies.

To verify this, additional simulation runs are performed in which only one source type is used, $\mathrm{PT}_G(5,5)$, but the shaping window is varied in the fashion of $W =$ 5, 10, 15, 20, 50, 100, 200. The results can be summarized as follows; no figure is necessary, since the output from the different cases is virtually identical. Increasing the window size from $W = 5,\ 10,\ 15$ caused an increase in maximum queue length of $Q = 268,\ 270,\ 272$, after which further increases in W have no more effect on Q. This is an expected result, since as explained above, as long as the shaping window can capture a burst and silence period of the cell stream, the shaper operates in State 1. Therefore, all else being equal, an increase in the shaping window size should have no effect on the shaped cell stream, as the results show. There is only the minor variation in *maximum* shaper queue size, which occurs due to the fact

that larger windows have the ability to shape larger bursts, since they now occur during State 1, whereas with a smaller window they would have occurred during State 3.

4.2.5.1 Comparison with Other Shapers

Due to the fact that the BOS is designed to shape specifically PT sources, it is not possible to compare the results here with any cited in the literature.

This section concludes the discussion and presentation of the Burst-oriented shaper. as well as this chapter pertaining to traffic shaping. The next and final chapter of this thesis reiterates the contributions of this work, as well as pointing the reader in a few directions along which the contributions can be extended.

Chapter 5

Contributions and Future Work

The integration of services is the driving force behind the design of the high speed data networks of today and tomorrow. These networks must be able to deliver a broad range of services and be capable of carrying diverse classes of traffic with very different source characteristics. In the case of ATM networks, the solution of these conflicting requirements is to negotiate a Traffic Contract at the UNI, which specifies a QoS level and the characteristics of the source. These characteristics are used by CAC and UPC to protect existing connections.

Unfortunately, the determination of source characteristics by either the user or network provider is difficult, or impossible in some situations. One method to characterize sources it to employ traffic shapers, which have the benefit of creating a more easily definable traffic stream at the cost of injecting delay into said stream. Thus, there is a need for a method that can accurately provide a description of traffic streams in a timely manner. Three contributions have been presented in this thesis in order a satisfy these needs.

The proposed traffic primitive classifier can be used to classify unknown traffic streams. This is accomplished by defining characteristics traffic streams, assuming they have undergone cellization at the AAL. The characteristics of the streams identified are collectively called traffic primitives, and are used to define training vectors in order for a neural network to learn the classification problem. The justification for using neural networks over conventional methods is due to their ability to generalize and their speed. Their generalization ability allows the training vectors to be simple and deterministic, and still allow the classification of probabilistic traffic streams. Their speed allows the classifications of the traffic primitive classifier to be useful at the control points of high speed networks, such as ATM.

The traffic classification results presented show that the neural networks not only can classify deterministic sources from which they are trained, that is DG, CBR, CBR-RC and PT sources, but also they can classify a wide range of random sources, such as the class of on-off sources. With the additional functionality of Traffic Primitive Histogram Identification and Stream Transition Tracking, the primitive classifier can be applied to characterize sources which are not on-off in nature. As well, the primitive classifier can be integrated into a policer to perform more complex policing actions, and to monitor traffic streams for a given set of occurrences. These contributions make the primitive classifier and its application useful at the source UNI, for CAC, UPC and shaping.

In addition to the traffic primitive classifier, two more contributions come in the form of two traffic shapers, the Minimized Variance shaper and the Burst-oriented shaper. Both shapers have the ability to produce near deterministic streams, given

appropriate sources are shaped, at fairly low costs in delay and buffer size at the shapers. In the case of the MVS, source information is utilized in order to find an optimal shaping parameter that has the effect of minimizing the interdeparture time variance of the shaped stream. In addition, two versions of the MVS are proposed, one that operates on traffic sources with known and analytic pdfs, and another which requires only an interarrival time histogram of the source.

On the other hand, the BOS does not require source information, since it assumes that bursts and silences emerge from the AAL, and so it attempts to spread a burst into the immediately following silence period. By doing so, it has the ability to define an unshaping parameter, which when embedded into the traffic stream, can be used to unshape the source at the destination UNI. This has the dual benefits of offering the network provider an ability to characterize sources and also improve network efficiency, but also to allow users to treat the network as a transparent connection. This is of importance to traffic sources which do not react well to shaping delays within their cell stream.

As far as future work is concerned, there are a multitude of directions in which this work can be extended and improved. In the case of primitive classification, it would be interesting to define a different set of traffic primitives, perhaps based on the requirements of an operating ATM network. It may be discovered that certain of the existing primitives are useful, but others should be deleted. As well, as noted from the results, while a underlying traffic pattern is valid, it should be included in another primitive class in order to improve the sensibility of a sequence of classifications. For the engine of the primitive classifier, the Backpropagation

neural network, more speed-up methods should be investigated in order to allow the training of larger networks. This would allow primitive classifiers with larger traffic windows to be implemented. In addition, the neural networks could be pruned, or be recurrent, or other methods could be employed to improve convergence and thus reduce training time. The functionality of the UNI can be combined into one control scheme, using neural networks, and thus the idea of Figure 1.6 in Section 1.4.1 should be explored. As for the applications of traffic classification, the two stated could be studied in more detail, and perhaps their feasibility studied, especially the primitive histogram comparing method. Also, other applications could be developed.

The MVS algorithm should be modified in order to take into account the overly pessimistic estimation of the maximum allowable increase in interdeparture time delay. Thus, the variance equation requires modification to take into account the cell arrivals that encounter the shaper with a non-empty queue.

In the case of the BOS, its robustness in terms of its unshaping parameter should be studied. It would be interesting to see how much the original and shaped cell stream differ when affected by loss or delay of the unshaping parameter, as well as delays of the cells of the shaped stream, that is cell delay variation. Additional goals of the BOS can be studied. For example, instead of shaping to a target cell spacing rate, the BOS could space cells so that the instantaneous rate of the stream observed in the shaping window is maintained. Also, heuristics should be added to bound the growth of the shaper queue, for example defining multiple shaping rates. In any case, further investigation should be performed to obtain an optimal size of

the shaping window over a wide range of traffic types.

With this, this thesis comes to a close. The author would like to thank the reader for the time taken to examine the results of many years of original research.

Appendix A

Neural Network & Backpropagation Primer

This appendix begins with a brief overview of multilayer feedforward neural networks. The advantages of neural networks over classical control schemes is sketched in Section A.1.1. Section A.1.2 introduces the basic building block of any neural network, the neuron, and Section A.1.3 describes the result of interconnecting many neurons, namely a neural network. Finally, the Appendix closes with the presentation of the Backpropagation algorithm in Section A.2.

A.1 Multilayer Feedforward Neural Networks

Neural networks, as their name implies, are a collection of entities called neurons, which are connected in a highly parallel manner. Both the neuron units and their interconnections, biologically referred to as synapses, are loosely based on their

counterparts in the human brain. The term "loosely" is used here, not because the neural connections in the brain are too simple, but rather because those connections are too complex! It is thought that the human brain contains on the order of 10^{11} neurons with 10^{15} synapses interconnecting them [Was89]. In fact, most of the neural network paradigms use only the simplest, most basic models of the neurons in the brain.

Despite the simplicity of the neuron model employed in the neural network field, some very impressive results have been achieved. Applications include speech and pattern recognition, weather forecasting, adaptive control, adaptive signal processing, expert systems, system identification, decision making, and many others. An extensive list of references can be found in [Hay94, Lip87, WL90]. These results are made possible due to the fact that neural networks consist of computational units which are connected in a massively parallel manner.

A.1.1 Classical and Neural Network Controllers

Neural networks have a number of important advantages over classical control schemes. Both perform an input mapping function, however the classical controller *specifies* the input-output relationship whereas the neural network *learns* it.

A.1.1.1 The Classical Controller

In classical control problems, where the mapping function may be very complicated and computationally expensive, especially when the number of inputs is large, a *real time* controller may not be realizable. In order to overcome the problems of real

time input mapping, classical controllers can calculate a *look-up table.* A look-up table consists of the set of input-output relationships that the controller maps. By calculating the required output for a given input off-line, computationally expensive control functions can be performed. This is best suited for binary inputs to the controller. However, if the number of inputs are large, or the inputs are real, then the look-up table will be prohibitively large. Therefore in many situations classical controllers are unsuitable.

A.1.1.2 A Neural Network as a Controller

A neural network is essentially an input transformation device. Figure A.1 shows a neural network as a "black box" that accepts a number of inputs and *maps* them

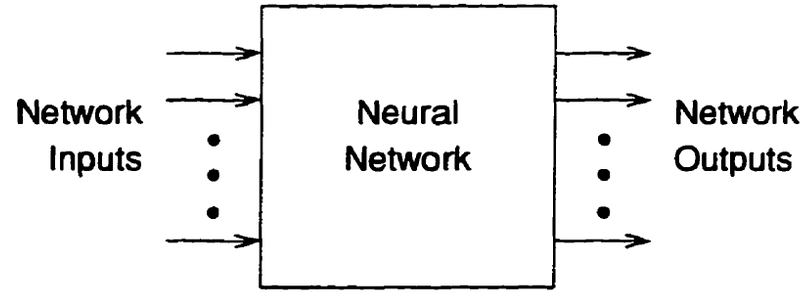


Figure A.1: The Neural Network Control Device

into a number of outputs. There is no theoretical constraint on the number of inputs that can map into a number of outputs. Inputs presented to the network are *fed forward* from one layer of neurons to the next. As the inputs traverse the network, they are *transformed* into the desired output mapping. This transformation is the

topic of Section A.1.3. However, before a neural network can map a given input into a desired output, it must learn the mapping function; when the network has learnt the mapping function, it is said to be *trained.* How a neural network is trained is the topic of Section A.2.

A.1.1.3 The Neural Network Advantage

There are two major reasons for choosing a neural network controller over a classical controller: parallelism and generalization. Neural networks, as mentioned earlier, are a collection of neurons connected in a massively *parallel* manner. They can perform the same computation for different inputs *simultaneously.* Due to their layered structure, these many computations are made available at the same time from one layer to the next. For conventional von Neumann computers, which compute *sequentially,* this type of problem is computationally expensive. As a result, the neural network structure can be much more efficient for some problems.

In addition to their parallelism, neural networks have the ability to *generalize.* Consider a situation in which a mapping function has been learnt by a neural network. If an input is presented to the network for which no output has been specified, the neural network acts to generalize the input to the nearest learnt input, and produces a well behaved interpolated or extrapolated output. On the other hand, while a classical controller may have this ability, it is more likely that some random, "uneducated" output will be produced. In addition, if the classical controller is implemented in software, then any unexpected inputs will most likely be treated the same way, or may cause the controller to fail. With this brief discussion of the advantages of neural networks over classical controllers complete,

the next section will describe the basic building block of a neural network.

A.1.2 The Neuron Model

The neuron is the basic building block of a neural network. Consider the neuron model of Figure A.2. It consists of a summation unit and a nonlinearity. The

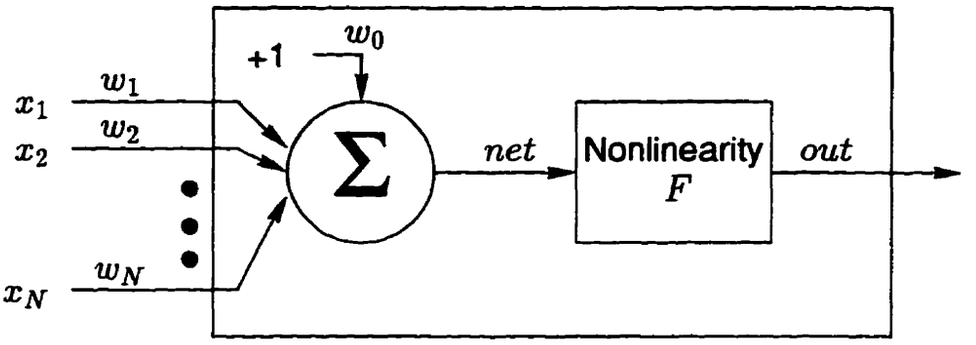


Figure A.2: A Simple Neuron Model

summation unit is the terminus for a number of network connections. The connections represent synapses, and each has an associated weight w_i.[1] The origin of each connection is a network input x_i.

The weights are variable, which allows neural networks to learn, or adapt. Accordingly, the summation block of Figure A.2 is sometimes referred to as an *adaptive linear combiner*. It is adaptive, since the weights are variable, and it is linear, since

[1]Note also that there is a bias "weight," w_0, which is considered to be a connection as well. Imagine it is connected to an input that is always "on."

it performs a sum. The sum that the neuron model performs is given by

$$net = \sum_{i=1}^{N} w_i x_i + w_0,$$

where it is assumed that there are N inputs to the combiner. Hence *net* represents the weighted sum of the network inputs to this particular neuron. This sum also can be represented very compactly in vector notation. If the inputs are given by the row vector[2]

$\mathbf{x}^T = [x_1 \; x_2 \; \cdots \; x_n]$ and the weights are given by the row vector $\mathbf{w}^T = [w_1 \; w_2 \; \cdots \; w_n]$, then the output of the combiner is

$$net = \mathbf{w}^T\mathbf{x} + w_0.$$

The neuron model of Figure A.2 also contains a nonlinearity, denoted by the function F. This function is usually referred to as an *activation function.* Figure A.3 depicts two popular activation functions, a threshold function and a sigmoidal function. The threshold function performs

$$out = \text{sgn}(net),$$

whereas the sigmoidal function performs

$$out = \frac{1}{1 + e^{-net}}. \tag{A.1}$$

[2]A vector **a** is assumed to be a column vector. Therefore, $\mathbf{a}^T$ is a row vector.

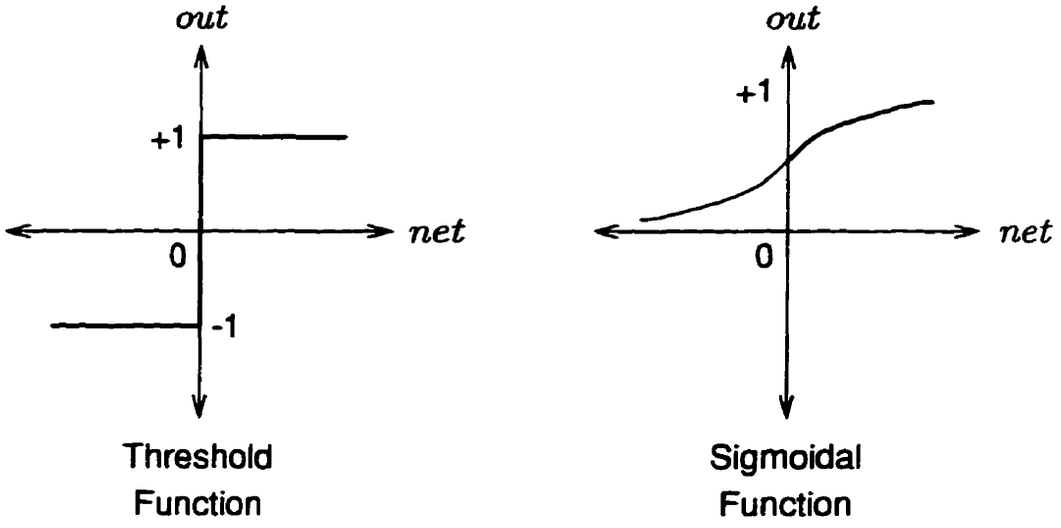


Figure A.3: Two Activation Functions

The form that the activation function takes, for the most part, determines the nature of the neuron. For example, if the activation function is a threshold, then the neuron is referred to as a *perceptron*.

Perceptron neural networks were studied as early as the 1940's. The networks consisted of just a single perceptron or just one layer of perceptrons. Initially, it was thought that perceptron networks would have a wide range of applicability, however as research continued they were found to be unsuitable for solving some very simple problems [MP88]. The major problem is that of *linear separability*. Since the perceptron performs a threshold on the weighted sum of its inputs, it is in fact separating its inputs with a straight line. If the inputs cannot be separated with a straight line, then the perceptron will not be able to learn the input-output relationships presented to it. Much work has been performed to overcome the

linear separability problem, such as mapping the inputs through nonlinearities and combining the output of two separate perceptrons (with the same inputs) [Was89, WL90], but this work leads to multilayered networks. Nonetheless, perceptrons are still an important topic in neural network research, and have laid the foundations for multilayered feedforward networks.

The sigmoidal[3], activation function of Figure A.3 has more utility than the threshold function for two reasons: it acts to scale the inputs, and it is continuously differentiable. If inputs to the network are both very small and very large, the very large inputs will "swamp" the very small. To overcome this problem a gain is required for the small inputs whereas no gain is required for the large inputs; the sigmoidal function provides this. Small inputs pass through the linear portion of the sigmoidal, while very large inputs are "clipped." The second reason for the sigmoidal activation function's utility is the fact that it is continuously differentiable. The derivative of the threshold function, on the other hand, contains an impulse. The importance of this observation will be made clear in Section A.2, where the Backpropagation method of neural network learning will be discussed.

A.1.3 The Neural Network

While a single neuron can perform some astonishing tasks, such as pattern recognition, the neuron model's real potential can be observed when many neurons are connected into networks. Once again, the motivation for multilayered networks is based on observations of their biological counterparts, the human brain. Until

[3]The sigmoidal function is also known as the logistic or squashing function in the literature.

recently, these networks were limited to a single layer of neurons because methods needed to train multilayered networks did not exist.

Consider the neural network of Figure A.4. This network is made up of one input layer and two layers of neurons, and can be referred to as a two layer network.[4]

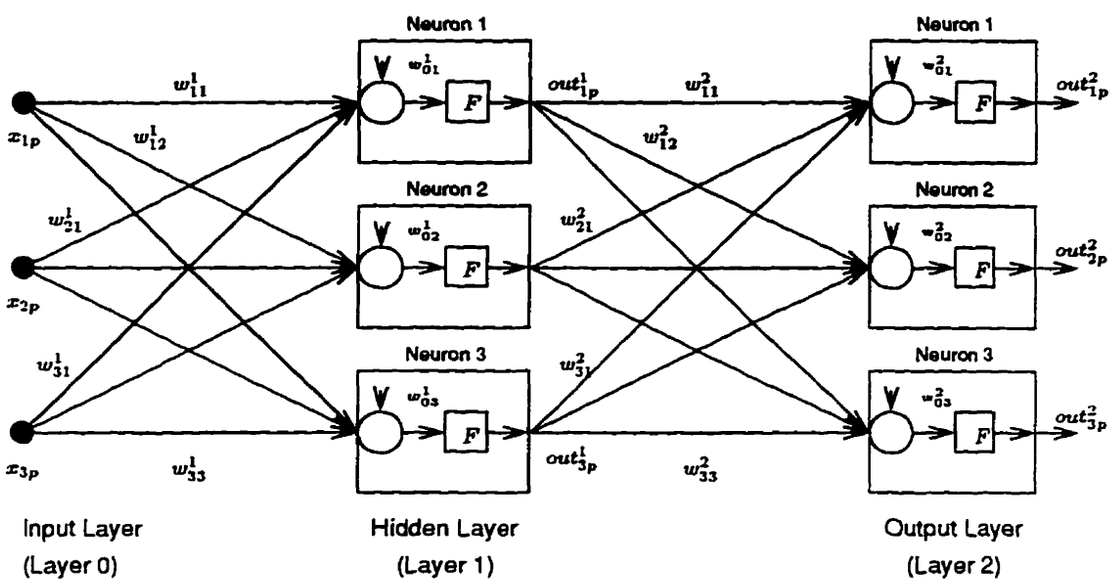


Figure A.4: A Two Layer Neural Network

The first layer, or *input layer*, contains three input units to which the inputs to the network are applied. The second layer, because it is "sandwiched" between the first and third layers, is called a *hidden layer*. In the network of Figure A.4, there is only one hidden layer containing three neurons. The third layer, or *output*

[4]Much of the literature would refer to the network of Figure A.4 as a *three* layer network. This would seem intuitively pleasing at first glance, however it does not correctly describe the network. The first layer does not contain any neurons at all, and so should not be considered in the naming convention. If the first layer is to be considered, it would be more correct to count the "layers of connections," which would still result in a two layer network.

layer, contains three output neurons. It is possible for a neural network to contain any number of hidden layers, however there is only one input layer and one output layer. Naturally, there is no constraint on the number of neurons in any layer or inputs to the network, nor are there constraints on the number of connections a neuron can terminate or initiate.

The input units are not really neurons. They are used to distribute the inputs to the network to various neurons in the network; they do not perform any neural computation, as described in Section A.1.2, of their own. Note also that the output neurons, while they do perform computation, do not initiate any connections to any neurons in the network.[5] Each layer in Figure A.4, except for the input layer, has a connection from every unit in the preceding layer. For example, output layer neuron one has a connection from each of hidden layer neurons one, two, and three. This special type of network has been aptly named a *fully connected* network. Due to the fact that there is very little theory on the number of connections required from one layer to another, or for that matter on the number of layers required or the number of neurons required in each layer, the fully connected network is widely used.[6] Each neuron in Figure A.4 has the exact same structure as the basic neural building block of Figure A.2, namely a summing unit with a bias value and a sigmoidal activation function. The *net* and *out* values are calculated in the same manner. However. since the weights can connect to more than one neuron, and since there is more

[5]If they did have connections back into the network, this would be called a *recurrent* neural network. If they had connections between each other, or any other neurons for that matter, then this network would contain *competition.*

[6]It turns out that, after training, connections between neurons that are not required will usually have a very low weight w_i, and so they could be deleted. The network resulting after the connection deletions must then be retrained.

than one set (layer) of weights, a new naming scheme must be developed. Label a general connection weight w_{ij}^{l}, where i represents the neuron at the origin of the connection, j represents the neuron at the terminus of the connection, and l represents the layer of the neuron at the terminus of the connection. As an example, the weight connecting neuron one in the hidden layer to neuron two in the output layer is denoted as w_{12}^{2}. In a similar manner, the quantities *net* and *out* are named according to their network layer and their specific neuron. However, since *net* and *out* also depend on the input vector $\mathbf{x}$, and since any useful network has more than one input vector, the naming scheme should be augmented to allow different input vectors. Assume a possible network input vector $\mathbf{x}$ is chosen from a finite set of P vectors. A particular input vector is denoted by $\mathbf{x}_p$, where $p = 1, 2, \ldots, P$. Therefore net_{ip}^{l} and out_{ip}^{l} denote the *net* and *out* values, respectively, for neuron i in layer l when the input vector $\mathbf{x}_p$ is presented to the network. The *net* and *out* values for the third neuron in the output layer when input vector $\mathbf{x}_p$ is presented, for example, are denoted as net_{3p}^{2} and out_{3p}^{2}, respectively. Some of these quantities are shown in Figure A.4.

A.1.3.1 An Example: A 3-3-3 Network

Figure A.4 is an example of a 3-3-3 neural network. The shorthand notation "3-3-3" indicates that this neural network has three input units, one hidden layer with three neurons, and three output neurons. It is understood that the network is fully connected. If a fully connected network under consideration had two hidden layers with six and four neurons, respectively, and with the same number of input units as above but only one output neuron, then its shorthand notation would be 3-6-4-1.

With the naming terminology completed, expressions for the output of the network can be obtained. Observing Figure A.4, the values *net* and *out* for the hidden layer are

$$\begin{aligned}
net^1_{1p} &= \sum_{i=1}^{3} w^1_{i1} x_{ip} + w^1_{01} \\
out^1_{1p} &= F(net^1_{1p}) \\
net^1_{2p} &= \sum_{i=1}^{3} w^1_{i2} x_{ip} + w^1_{02} \\
out^1_{2p} &= F(net^1_{2p}) \\
net^1_{3p} &= \sum_{i=1}^{3} w^1_{i3} x_{ip} + w^1_{03} \\
out^1_{3p} &= F(net^1_{3p}),
\end{aligned}$$

and the *net* and *out* values for the output layer are

$$\begin{aligned}
net^2_{1p} &= \sum_{i=1}^{3} w^1_{i1} out^1_{ip} + w^2_{01} \\
out^2_{1p} &= F(net^2_{1p}) \\
net^2_{2p} &= \sum_{i=1}^{3} w^1_{i2} out^1_{ip} + w^2_{02} \\
out^2_{2p} &= F(net^2_{2p}) \\
net^2_{3p} &= \sum_{i=1}^{3} w^1_{i3} out^1_{ip} + w^2_{03} \\
out^2_{3p} &= F(net^2_{3p}),
\end{aligned}$$

where w^l_{0j} represents the bias value for neuron j of layer l, p ranges over all possible input vectors, and F is the sigmoidal activation function given in Section A.1.2.

Again, as in Section A.1.2, these sums can be conveniently written in vector notation. Referring to Figure A.4, the row vectors of the weights that terminate on neurons one, two and three in the hidden layer are

$$\begin{aligned} \mathbf{w}_1^{1T} &= [w_{11}^1\ w_{21}^1\ w_{31}^1] \\ \mathbf{w}_2^{1T} &= [w_{12}^1\ w_{22}^1\ w_{32}^1] \\ \mathbf{w}_3^{1T} &= [w_{13}^1\ w_{23}^1\ w_{33}^1], \end{aligned}$$

respectively. A weight matrix of these three weight vectors can be constructed by

$$\begin{aligned} \mathbf{W}^1 \equiv \begin{bmatrix} \mathbf{w}_1^{1T} \\ \mathbf{w}_2^{1T} \\ \mathbf{w}_3^{1T} \end{bmatrix}^T &= \begin{bmatrix} w_{11}^1 & w_{21}^1 & w_{31}^1 \\ w_{12}^1 & w_{22}^1 & w_{32}^1 \\ w_{13}^1 & w_{23}^1 & w_{33}^1 \end{bmatrix}^T \\ &= \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \end{bmatrix}. \end{aligned}$$

Using a similar construction, the weight matrix for the output layer neurons is

$$\mathbf{W}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \end{bmatrix}.$$

The neuron biases can also be written in vector notation as

$$\Theta^1 = \begin{bmatrix} w_{01}^1 \\ w_{02}^1 \\ w_{03}^1 \end{bmatrix}, \qquad \Theta^2 = \begin{bmatrix} w_{01}^2 \\ w_{02}^2 \\ w_{03}^2 \end{bmatrix}.$$

Finally, the **net** and **out** vectors for each layer of neurons in Figure A.4, and for each input vector $\mathbf{x}_p$, can be written

$$\begin{aligned} \mathbf{net}_p^{1^T} &= [net_{1p}^1\ net_{2p}^1\ net_{3p}^1] \\ \mathbf{out}_p^{1^T} &= [out_{1p}^1\ out_{2p}^1\ out_{3p}^1] \\ \mathbf{net}_p^{2^T} &= [net_{1p}^2\ net_{2p}^2\ net_{3p}^2] \\ \mathbf{out}_p^{2^T} &= [out_{1p}^2\ out_{2p}^2\ out_{3p}^2], \end{aligned}$$

where

$$\begin{aligned} \mathbf{net}_p^1 &= \mathbf{W}^{1^T}\mathbf{x}_p + \Theta^1 \\ \mathbf{out}_p^1 &= F(\mathbf{net}_p^1) \\ \mathbf{net}_p^2 &= \mathbf{W}^{2^T}\mathbf{out}_p^1 + \Theta^2 \\ \mathbf{out}_p^2 &= F(\mathbf{net}_p^2). \end{aligned}$$

A.1.3.2 The General Network: N^0-((N_{M-1}^l))-N^M

The expressions for *net* and *out* of Section A.1.3.1 can be applied to any general neural network. Consider a network that has M layers of *neurons*, where the

input layer is layer $l = 0$, there are $M - 1$ hidden layers, and the output layer is layer $l = M$. Each layer consists of N^l neurons, where N^0 denotes the number of *inputs* to the network, *not* neurons. Thus, the shorthand notation for a fully connected general network is N^0-$((N^l_{M-1}))$-N^M, where N^0 represents the number of input units, $((N^l_{M-1}))$ represents the number of neurons in each of the $M - 1$ hidden layers, and N^M represents the number of output layer neurons. Now, if the network inputs x_{ip} are renamed out^0_{ip}, then general *net* and *out* equations can be stated as

$$\begin{aligned} net^l_{jp} &= \sum_{i=1}^{N^{l-1}} w^l_{ij} out^{l-1}_{ip} + w^l_{0j} \\ out^l_{jp} &= F(net^l_{jp}) \end{aligned} \qquad \begin{cases} l = 1, 2, \ldots, M \\ j = 1, 2, \ldots, N^l \\ p = 1, 2, \ldots, P. \end{cases} \tag{A.2}$$

In general, the weight matrix $\mathbf{W}^l$ for layer l is of the order $N^{l-1} \times N^l$, the $\mathbf{net}^l_p$, $\mathbf{out}^l_p$ and Θ^l vectors are of the order N^l. The following iterative matrix equations represent the operation of the neural network:

$$\begin{aligned} \mathbf{net}^l_p &= \mathbf{W}^{l^T} \mathbf{out}^{l-1}_p + \Theta^l \\ \mathbf{out}^l_p &= F(\mathbf{net}^l_p) \end{aligned} \qquad \begin{cases} l = 1, 2, \ldots, M \\ p = 1, 2, \ldots, P, \end{cases} \tag{A.3}$$

where

$$\mathbf{net}_p^l = \begin{bmatrix} net_{1p}^l \\ net_{2p}^l \\ \vdots \\ net_{N^l p}^l \end{bmatrix}, \qquad \mathbf{out}_p^l = \begin{bmatrix} out_{1p}^l \\ out_{2p}^l \\ \vdots \\ out_{N^l p}^l \end{bmatrix}, \qquad \Theta^l = \begin{bmatrix} w_{01}^l \\ w_{02}^l \\ \vdots \\ w_{0N^l}^l \end{bmatrix},$$

and

$$\mathbf{W}^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots & w_{1N^l}^l \\ w_{21}^l & w_{22}^l & \cdots & w_{2N^l}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{N^{l-1}1}^l & w_{N^{l-1}2}^l & \cdots & w_{N^{l-1}N^l}^l \end{bmatrix}.$$

Observing the set of matrix Equations A.3, it can be seen that the output of neurons in a given layer l can be calculated only after the output of neurons in layer $l-1$ has been determined.[7] In this sense, the output of the previous layer *feeds* the inputs of the current layer, and hence the term *feedforward* neural network.

This concludes the discussion of how inputs are transformed into outputs in a fully connected feedforward neural network. Equation (A.3) gives the iterative matrix equations to perform the required mapping function. The next section describes how the connection weights w_{ij}^l can be varied so that the neural network learns the required input-output relationships.

---

[7]Note that $\mathbf{out}_p^0 \equiv \mathbf{x}_p = [x_{1p}\ x_{2p}\ \cdots\ x_{N^0 p}]^T$.

A.2 The Backpropagation Algorithm

Section A.1.3 described how inputs propagate from one layer of neurons to the next, and finally arrive as outputs. The connection weights of the various layers in the network perform the transformations required to map a given input into a desired output, as was discussed in Section A.1.1. In this section, the Backpropagation algorithm required to "teach" the mapping function to the neural network is presented. The Backpropagation algorithm for multilayered feedforward networks with sigmoidal activation functions is a relatively new development [RHW86]. It involves presenting an input to the network, observing the output the network produces, comparing this output to the desired output, and then modifying the connection weights w_{ij}^{l} in such a way as to minimize the *squared* difference between the network output and the desired output. The following presentation of the Backpropagation algorithm follows that given in [Hay94, RHW86].

A.2.1 Statement of the Algorithm

The Backpropagation learning rule is a generalization of the Delta Rule or the Widrow-Hoff Rule [WL90]. The delta rule is a simplification of the classical Hebbian learning rule. Using the same notation and definitions as in Section A.1.3, the

Backpropagation learning rule can be stated as

$$\Delta_p w_{ij}^l(n) = \eta \delta_{jp}^l(n) \left[1 + (1 - \Xi(i)) \left(out_{ip}^{l-1}(n) - 1\right)\right] \begin{cases} i = 0, 1, \ldots, N^{l-1} \\ j = 1, 2, \ldots, N^l \\ l = 1, 2, \ldots, M \\ p = 1, 2, \ldots, P \\ n = 1, 2, \ldots, \end{cases} \quad \text{(A.4)}$$

$$\Xi(i) = \begin{cases} 1 & i = 0 \\ 0 & \text{otherwise,} \end{cases}$$

where $\Delta_p w_{ij}^l(n)$ is the change in connection weight from neuron i in the previous layer to neuron j in layer l, η is the *learning rate*, $\delta_{jp}^l(n)$ is the *instantaneous square error derivative* of the network output error with respect to the *net* value of neuron j in layer l, and $out_{ip}^{l-1}(n)$ is the output of neuron i in layer $l-1$. The index n represents the iteration step of the algorithm; after all the network weights have been updated, n is incremented.

The learning rate η is simply a constant coefficient which determines how fast the neural network will learn the input-output relationships presented. If η is large, then the network will learn quickly, but there will be a higher chance that the weight changes oscillate. If η is small, the chance of weight oscillations will be decreased, however the training time will increase. Typically, $0.1 \leq \eta \leq 1.0$. The squared error derivative, δ, is calculated differently depending on whether the neuron in question is an output layer neuron or a hidden layer neuron. For an output layer

neuron, $l = M$,

$$\delta_{jp}^{M}(n) = \left(t_{jp} - out_{jp}^{M}(n)\right) F'\left(net_{jp}^{M}(n)\right) \quad \begin{cases} j &= 1, 2, \ldots, N^M \\ p &= 1, 2, \ldots, P \\ n &= 1, 2, \ldots, \end{cases} \tag{A.5}$$

and for a hidden layer l neuron,

$$\delta_{jp}^{l}(n) = \sum_{k=1}^{N^{l+1}} \delta_{kp}^{l+1}(n) w_{jk}^{l+1}(n) F'\left(net_{jp}^{l}(n)\right) \quad \begin{cases} j &= 1, 2, \ldots, N^l \\ l &= M-1, M-2, \ldots, 2, 1 \\ p &= 1, 2, \ldots, P \\ n &= 1, 2, \ldots . \end{cases} \tag{A.6}$$

In Equation (A.5) for the output layer, t_{jp} represents the desired network output of neuron j in the output layer when the input to the network is $\mathbf{x}_p$. In both Equations (A.5) and (A.6), F' denotes the derivative of the activation function F with respect to the appropriate *net* value. Also note that the quantities $net_{jp}^{l}(n)$ and $out_{jp}^{l}(n)$ are calculated with the same method given by Equation (A.2); their dependency on n simply reflects the fact that they are calculated using weights $w_{ij}^{l}(n)$ which are updated after every iteration.

The bounds on l in Equation (A.6) stress the fact that the weight changes specified by the Backpropagation algorithm start from the output layer and *back-propagate* through the network to the first hidden layer. In fact, this was the major breakthrough that allowed multilayered feedforward neural networks to be trained. The error observed at the output layer is used to train not only the weights ter-

minating on the output layer, but also all other weights in the network. This is accomplished by *propagating* the squared error derivatives calculated at each output neuron *back* through the connection weights to all neurons in the network.

With the learning rule defined, which specifies the changes to the weights, the Backpropagation algorithm can be stated

$$w_{ij}^{l}(n+1) = w_{ij}^{l}(n) + \Delta_p w_{ij}^{l}(n) \qquad \begin{cases} i &= 0,\ 1,\ \ldots,\ N^{l-1} \\ j &= 1,\ 2,\ \ldots,\ N^{l} \\ l &= 1,\ 2,\ \ldots,\ M \\ p &= 1,\ 2,\ \ldots,\ P \\ n &= 1,\ 2,\ \ldots\ . \end{cases} \tag{A.7}$$

In the above, the iteration step n is incremented after all weights w_{ij}^{l} in the network have been updated. Two methods of updating network weights are generally used in the literature. The first method simply updates all the network weights after a given input vector $\mathbf{x}_p$ has been presented to the network. This is the method that Equation (A.7) describes. A second method involves summing all the weight updates calculated for each input vector, and then applying a single update to the network weights after the entire training set has been presented. In this case, the Backpropagation algorithm is modified to be

$$w_{ij}^{l}(n+1) = w_{ij}^{l}(n) + \sum_{p=1}^{P} \Delta_p w_{ij}^{l}(n) \qquad \begin{cases} i &= 0,\ 1,\ \ldots,\ N^{l-1} \\ j &= 1,\ 2,\ \ldots,\ N^{l} \\ l &= 1,\ 2,\ \ldots,\ M \\ n &= 1,\ 2,\ \ldots\ . \end{cases} \tag{A.8}$$

While Equation (A.8) is closer to the true gradient descent method, it is seldom used in practice, especially if the training set P is large. The assumption that the error surface E in weight space is linear in the area of w_{ij}^l is justified if the learning rate η is small. As a result, Equation (A.7) is preferred over Equation (A.8). The following section verifies that the Backpropagation of the squared error derivatives, through the connection weights, minimize the output error in a sum square sense.

A.2.2 Derivation of the Algorithm

The Backpropagation algorithm is essentially a gradient descent method in connection weight space. It acts to minimize the overall network output error, E, which is defined over all input vectors $\mathbf{x}_p$ as

$$E(n) = \sum_{p=1}^{P} E_p(n).$$

The output error of the network for a given input vector $\mathbf{x}_p$ is defined as

$$E_p(n) = \frac{1}{2} \sum_{j=1}^{N^M} \left(t_{jp} - out_{jp}^M(n)\right)^2, \tag{A.9}$$

for all $p = 1, 2, \ldots, P$ and $n = 1, 2, \ldots$. The quantity t_{jp} represents the desired output of neuron j in the output layer when the input to the network is $\mathbf{x}_p$. If the values of t_{jp} for all output neurons are assembled into a vector, this vector is referred to as the *training vector* $\mathbf{t}_p$, where $\mathbf{t}_p = [t_{1p}\, t_{2p} \cdots t_{N^M p}]^T$. Hence, a given input vector $\mathbf{x}_p$ has a corresponding training vector $\mathbf{t}_p$. Together, these two vectors make up a *training pair*. As a result, to train a network to learn P input-output

relationships requires P training pairs, which are called the *training set.*[8] Therefore, $E_p(n)$ represents the *sum square error* of the output of the network with respect to a particular training pair p, whereas $E(n)$ represents the sum square error of the output of the network over *all* training pairs. The following two sections will show that the Backpropagation algorithm performs a gradient descent in $E(n)$; that is the overall network output error will be minimized with respect to the weights in the network.

In order to show that the Backpropagation algorithm (generalized delta rule) minimizes the output error of the network with respect to the network weights in a sum square sense, it must be shown that the derivative of the overall network error $E(n)$ is proportional to the weight changes specified by the Backpropagation algorithm of Equation (A.4), with a negative coefficient of proportionality. Since

$$\begin{aligned} \frac{\partial E(n)}{\partial w_{ij}^l} &= \frac{\partial}{\partial w_{ij}^l}\left(\sum_{p=1}^{P} E_p(n)\right) \\ &= \sum_{p=1}^{P} \frac{\partial E_p(n)}{\partial w_{ij}^l}, \end{aligned}$$

minimizing $E_p(n)$ minimizes $\sum_{p=1}^{P} E_p(n)$ which is equivalent to minimizing $E(n)$. Therefore, it must be shown that

$$\Delta_p w_{ij}^l(n) \propto -\frac{\partial E_p(n)}{\partial w_{ij}^l(n)},$$

---

[8] In fact, more than P training pairs are required. After the network has learnt the P input-output relationships, the training is usually validated using another set of V training pairs, $\hat{\mathbf{x}}_v$ and $\mathbf{v}_v$, $v = 1, 2, \ldots, V$, which is referred to as the *validation set.*

for all $p = 1, 2, \ldots, P$. Due to the fact that hidden layer neurons do not obtain direct feedback of the output error $E_p(n)$, two measures of the instantaneous squared error derivative δ result: one for the output layer neurons, and one for the hidden layer neurons.

A.2.2.1 δ for an Output Layer Neuron

To show[9] that the Backpropagation algorithm minimizes the network output error with respect to the weights that terminate on the output layer neurons, that is $l = M$, we note that when attempting to calculate $\frac{\partial E_p}{\partial w_{ij}^M}$ the error E_p does not depend directly on the weights w_{ij}^M. Invoking the chain rule results in

$$\frac{\partial E_p}{\partial w_{ij}^M} = \frac{\partial E_p}{\partial out_{jp}^M} \frac{\partial out_{jp}^M}{\partial w_{ij}^M}.$$

This gives an expression for the change in network error with respect to changes in the weights terminating on neurons in the output layer, in terms of the change in network error with respect to changes in the output of neurons in the output layer and the change of the output of neurons in the output layer with respect to changes in the weights terminating on neurons in the output layer. Unfortunately, out_{jp}^M does not depend directly on w_{ij}^M, but it does depend directly on net_{jp}^M. Using the chain rule once more gives

$$\frac{\partial out_{jp}^M}{\partial w_{ij}^M} = \frac{\partial out_{jp}^M}{\partial net_{jp}^M} \frac{\partial net_{jp}^M}{\partial w_{ij}^M},$$

[9]For clarity, in this and the following section the dependence of the variables on the iteration step n will be suppressed. Since the form of the Backpropagation learning rule does not depend on the iteration step n, there is no loss of generality.

and so

$$\frac{\partial E_p}{\partial w_{ij}^M} = \frac{\partial E_p}{\partial out_{jp}^M}\frac{\partial out_{jp}^M}{\partial net_{jp}^M}\frac{\partial net_{jp}^M}{\partial w_{ij}^M} \qquad \begin{cases} i &= 0,\ 1,\ \ldots,\ N^{M-1} \\ j &= 1,\ 2,\ \ldots,\ N^M \\ p &= 1,\ 2,\ \ldots,\ P. \end{cases} \tag{A.10}$$

Using Equation (A.9) the first partial derivative on the right hand side of Equation (A.10) can be calculated as

$$\begin{aligned} \frac{\partial E_p}{\partial out_{jp}^M} &= \frac{\partial}{\partial out_{jp}^M}\left(\frac{1}{2}\sum_{j=1}^{N^M}\left(t_{jp} - out_{jp}^M\right)^2\right) \\ &= \frac{1}{2}(2)(-1)(t_{jp} - out_{jp}^M) \\ &= -(t_{jp} - out_{jp}^M), \end{aligned}$$

for $j = 1,\ 2,\ \ldots,\ N^M$ and $p = 1,\ 2,\ \ldots,\ P$. Using Equation (A.1) and the expression for out_{jp}^M in Equation (A.2) the second partial derivative can be calculated as

$$\begin{aligned} \frac{\partial out_{jp}^M}{\partial net_{jp}^M} &= \frac{\partial}{\partial net_{jp}^M}\left(F(net_{jp}^M)\right) \\ &= \frac{\partial}{\partial net_{jp}^M}\left(\left(1 + e^{-net_{jp}^M}\right)^{-1}\right) \\ &= \frac{e^{-net_{jp}^M}}{\left(1 + e^{-net_{jp}^M}\right)^2} \\ &= out_{jp}^M(1 - out_{jp}^M) \\ &= F'(net_{jp}^M), \end{aligned}$$

for $j = 1,\ 2,\ \ldots,\ N^M$ and $p = 1,\ 2,\ \ldots,\ P$, and where the following relationships were used

$$\begin{aligned} out_{jp}^M &= F(net_{jp}^M) \\ &= \frac{1}{1+e^{-net_{jp}^M}}, \\ (out_{jp}^M)^{-2} &= (1+e^{-net_{jp}^M})^2, \\ \frac{1-out_{jp}^M}{out_{jp}^M} &= e^{-net_{jp}^M}. \end{aligned}$$

As mentioned in Section A.1.2, it can be seen that for F' to exist, F must be continuously differentiable. This contributed to the reasons why the Backpropagation algorithm was not developed earlier. In the 1960's, almost all neural network research dealt with neurons with the threshold activation function of Figure A.3, which is not continuously differentiable.

Lastly, using the expression for net_{jp}^M in Equation (A.2), and the definitions of Equation (A.3), the third partial derivative of Equation (A.10) can be calculated as

$$\begin{aligned} \frac{\partial net_{jp}^M}{\partial w_{ij}^M} &= \frac{\partial}{\partial w_{ij}^M}\left(\sum_{k=0}^{N^{M-1}} w_{kj}^M out_{kp}^{M-1} + w_{0j}^M\right) \\ &= \frac{\partial w_{ij}^M}{\partial w_{ij}^M} out_{ip}^{M-1} + \frac{\partial w_{0j}^M}{\partial w_{ij}^M} \\ &= 1 + (1-\Xi(i))\left(out_{ip}^{M-1} - 1\right), \end{aligned}$$

for $i = 0,\ 1,\ \ldots,\ N^{M-1}$, $j = 1,\ 2,\ \ldots,\ N^M$, and $p = 1,\ 2,\ \ldots,\ P$. Bringing the expressions for the partial derivatives of the right hand side of Equation (A.10)

together gives

$$\frac{\partial E_p}{\partial w_{ij}^M} = -(t_{jp} - out_{jp}^M) F'(net_{jp}^M) \left[1 + (1 - \Xi(i)) \left(out_{ip}^{M-1} - 1\right)\right]$$

$$\begin{cases} i &= 0, 1, \ldots, N^{M-1} \\ j &= 1, 2, \ldots, N^M \\ p &= 1, 2, \ldots, P. \end{cases}$$

Now, for convenience, define

$$\delta_{jp}^M \equiv (t_{jp} - out_{jp}^M) F'(net_{jp}^M),$$

which is equivalent to

$$\delta_{jp}^M = -\frac{\partial E_p}{\partial out_{jp}^M} \frac{\partial out_{jp}^M}{\partial net_{jp}^M} = -\frac{\partial E_p}{\partial net_{jp}^M},$$

and is called the instantaneous squared error derivative, for neurons in the output layer, as was introduced in Section A.2.1. This gives

$$\frac{\partial E_p}{\partial w_{ij}^M} = -\delta_{jp}^M \left[1 + (1 - \Xi(i)) \left(out_{ip}^{M-1} - 1\right)\right],$$

or

$$\delta_{jp}^M \left[1 + (1 - \Xi(i)) \left(out_{ip}^{M-1} - 1\right)\right] = -\frac{\partial E_p}{\partial w_{ij}^M},$$

for $i = 0,\ 1,\ \ldots,\ N^{M-1}$. Therefore, this shows that

$$\Delta_p w_{ij}^M \propto -\frac{\partial E_p}{\partial w_{ij}^M},$$

and so finally

$$\Delta_p w_{ij}^M = \eta \delta_{jp}^M \left[1 + (1 - \Xi(i)) \left(out_{ip}^{M-1} - 1\right)\right] \quad \begin{cases} i &= 0,\ 1,\ \ldots,\ N^{M-1} \\ j &= 1,\ 2,\ \ldots,\ N^M \\ p &= 1,\ 2,\ \ldots,\ P, \end{cases}$$

where η is a constant coefficient of proportionality.

This completes the verification of the Backpropagation algorithm for the neurons in the output layer, that is $l = M$. The next section will briefly repeat this verification for the remainder of the neurons in the network, namely those in hidden layers $l = 1,\ 2,\ \ldots,\ M - 1$.

A.2.2.2 δ for a Hidden Layer Neuron

The verification of the Backpropagation algorithm for the hidden layer neurons is similar to that for the output layer neurons. Reconsider the equation

$$\frac{\partial E_p}{\partial w_{ij}^M} = \frac{\partial E_p}{\partial out_{jp}^M} \frac{\partial out_{jp}^M}{\partial w_{ij}^M},$$

which gives an expression for the change in network error with respect to changes in the weights terminating on the neurons in the output layer. Also, refer to neuron one in the hidden layer of Figure A.4. That is, consider the second to last layer in

the network, the last hidden layer, layer $l = M - 1$. The error E_p does not depend directly on w_{ij}^{M-1} (w_{ij}^{1} in Figure A.4) since the weights w_{ij}^{M-1} that terminate on the hidden layer are not connected to the output layer. However, E_p *does* depend directly on out_{jp}^{M-1}, $j = 1, 2, \ldots, N^{M-1}$, since the outputs of the hidden layer neurons affect the outputs of the network, namely out_{kp}^{M}, $k = 1, 2, \ldots, N^{M}$.

Rewriting the above partial differential equation for weights in the last hidden layer gives

$$\begin{aligned}
\frac{\partial E_p}{\partial w_{ij}^{M-1}} &= \frac{\partial E_p}{\partial out_{jp}^{M}} \frac{\partial out_{jp}^{M}}{\partial w_{ij}^{M-1}} \\
&= \frac{\partial E_p}{\partial out_{jp}^{M}} \frac{\partial out_{jp}^{M}}{\partial out_{jp}^{M-1}} \frac{\partial out_{jp}^{M-1}}{\partial w_{ij}^{M-1}} \\
&= \frac{\partial E_p}{\partial out_{jp}^{M-1}} \frac{\partial out_{jp}^{M-1}}{\partial w_{ij}^{M-1}}.
\end{aligned}$$

The partial derivative $\frac{\partial E_p}{\partial out_{jp}^{M-1}}$ represents the change in error given by a change in the output of neuron j in the hidden layer, and this output is affected by changes to the weights that neuron j terminates. That is, changes in w_{ij}^{M-1} will affect out_{jp}^{M-1}. But since out_{jp}^{M-1} affects the *net* values to, and thus the *out* values from, every neuron in the output layer through the weights w_{ij}^{M}, it can be seen that $\frac{\partial E_p}{\partial out_{jp}^{M-1}}$ must take into consideration all of the neurons in the output layer. Expressing this mathematically,

$$\frac{\partial E_p}{\partial out_{jp}^{M-1}} = \sum_{k=1}^{N^M} \frac{\partial E_p}{\partial net_{kp}^{M}} \frac{\partial net_{kp}^{M}}{\partial out_{jp}^{M-1}}.$$

Substituting this into the above equation gives

$$\begin{aligned}
\frac{\partial E_p}{\partial w_{ij}^{M-1}} &= \sum_{k=1}^{N^M} \frac{\partial E_p}{\partial net_{kp}^M} \frac{\partial net_{kp}^M}{\partial out_{jp}^{M-1}} \frac{\partial out_{jp}^{M-1}}{\partial w_{ij}^{M-1}} \\
&= \sum_{k=1}^{N^M} \frac{\partial E_p}{\partial net_{kp}^M} \frac{\partial}{\partial out_{jp}^{M-1}} \left(\sum_{n=1}^{N^{M-1}} w_{nk}^M out_{np}^{M-1} + w_{0k}^M \right) \frac{\partial out_{jp}^{M-1}}{\partial net_{jp}^{M-1}} \frac{\partial net_{jp}^{M-1}}{\partial w_{ij}^{M-1}} \\
&= \sum_{k=1}^{N^M} \frac{\partial E_p}{\partial net_{kp}^M} w_{jk}^M F'(net_{jp}^{M-1}) \left[1 + (1 - \Xi(i)) \left(out_{ip}^{M-2} - 1 \right) \right],
\end{aligned}$$

where $i = 0,\ 1,\ \ldots,\ N^{M-2}$, and where some relations of Section A.2.2.1 were used to arrive at the last line.

Now notice that the outputs out_{jp}^{M-1} of the hidden layer depend directly on the output of the previous hidden layer out_{np}^{M-2} through the weights w_{ij}^{M-1}. Therefore, through the weights a recursive relationship can be formed. In the general case, for hidden layer l

$$\frac{\partial E_p}{\partial w_{ij}^l} = \sum_{k=1}^{N^{l+1}} \frac{\partial E_p}{\partial net_{kp}^{l+1}} w_{jk}^{l+1} F'(net_{jp}^l) \left[1 + (1 - \Xi(i)) \left(out_{ip}^{l-1} - 1 \right) \right]$$

$$\begin{cases}
i &= 0,\ 1,\ \ldots,\ N^{l-1} \\
j &= 1,\ 2,\ \ldots,\ N^l \\
l &= M-1,\ M-2,\ \ldots,\ 2,\ 1 \\
p &= 1,\ 2,\ \ldots,\ P,
\end{cases} \tag{A.11}$$

In Section A.2.2.1, the definition for the instantaneous squared error derivative of neurons in the output layer, $\delta_{jp}^M = -\frac{\partial E_p}{\partial net_{jp}^M}$, was made. In a similar vein, δ_{jp}^l,

the instantaneous squared error derivative for neurons in the hidden layers can be defined as

$$\delta_{jp}^{l} = -\frac{\partial E_p}{\partial net_{jp}^{l}}.$$

Thus, Equation (A.11) becomes

$$\frac{\partial E_p}{\partial w_{ij}^{l}} = -\sum_{k=1}^{N^{l+1}} \delta_{kp}^{l+1} w_{jk}^{l+1} F'(net_{jp}^{l}) \left[1 + (1 - \Xi(i)) \left(out_{ip}^{l-1} - 1\right)\right],$$

for $i = 1, 2, \ldots, N^{l-1}$. Notice that

$$\delta_{jp}^{l} = -\frac{\partial E_p}{\partial net_{jp}^{l}} = -\frac{\partial E_p}{\partial out_{jp}^{l}} \frac{\partial out_{jp}^{l}}{\partial net_{jp}^{l}},$$

so that a recursive equation in δ can be formed by

$$\delta_{jp}^{l} = \sum_{k=1}^{N^{l+1}} \delta_{kp}^{l+1} w_{jk}^{l+1} F'(net_{jp}^{l}).$$

Observing this last equation, it can be seen that the instantaneous squared error derivatives for neurons in a hidden layer depend on the instantaneous squared error derivative for neurons in subsequent hidden layers, and ultimately of the instantaneous squared error derivatives for neurons in the output layer. To conclude the verification, as in Section A.2.2.1,

$$\frac{\partial E_p}{\partial w_{ij}^{l}} = -\delta_{jp}^{l} \left[1 + (1 - \Xi(i)) \left(out_{ip}^{l-1} - 1\right)\right],$$

or

$$\delta_{jp}^{l}\left[1+(1-\Xi(i))\left(out_{ip}^{l-1}-1\right)\right]=-\frac{\partial E_p}{\partial w_{ij}^{l}}$$

for $i = 1, 2, \ldots, N^{l-1}$, and including a constant coefficient of proportionality, η, completes the verification,

$$\Delta_p w_{ij}^{l}=\eta\delta_{jp}^{l}\left[1+(1-\Xi(i))\left(out_{ip}^{l-1}-1\right)\right] \quad \begin{cases} i = 0, 1, \ldots, N^{l-1} \\ j = 1, 2, \ldots, N^{l} \\ l = M-1, M-2, \ldots, 2, 1 \\ p = 1, 2, \ldots, P. \end{cases}$$

Appendix B

Neural Network Training Methodology

This appendix summarizes the methodology used to train the three neural networks that implement the primitive classifier of Chapter 2. The method used to determine the number of hidden layers and the number of neurons in each hidden layer is heuristic, and strongly based on experience gained from training smaller sized neural networks than those eventually used. However, some guidance is provided in the literature, for example [Guy91]. The reason for starting with relatively small neural networks is that, besides the obvious reason of requiring less time to train, it is desirable in order to avoid problems of overtraining. Hence, the smaller the neural network the better.

Unlike the usual neural network training methods where the training vectors can be divided into a training set and a validation set, this cannot be performed here. Once the neural network is trained so that it has a sufficiently low classification

error rate on its training vectors, it cannot be presented with more PT vectors, for example, since all the possible PT training vectors are specified in the training set. This is due to the fact the the primitive classifier is designed to detect traffic primitives of traffic streams from the probabilistic partition, as discussed in Section 2.1.2. Hence, for the three neural networks introduced in Section 2.2, they are verified using the training set in Section 2.3, but the real test is how they perform with unknown, probabilistic sources in Section 2.4.

The following section gives some results of attempting to train a few neural networks, from which experience is gained with the training problem at hand. The results of training neural networks of various sizes is provided in Table B.1. Section B.2 develops a heuristic based on this experience which can be used to estimate the number of neurons required for convergence in the larger neural networks which are required to learn the traffic classification problem. Finally, Section B.3 concludes this appendix with Table B.2, which shows the computer time involved in training the three neural networks of Chapter 2.

B.1 Some Training Examples

In the following examples, the number of inputs and outputs of the neural network are ten and nine, respectively, and the training vectors given in Tables C.1–C.4 of Appendix C are used. In addition, one of the criterion for deciding if a neural network is trained sufficiently is the value of the Mean Squared Error (MSE) over its training set. The expression for calculating the MSE in Backpropagation neural networks is given by Equation (A.9) in Section A.2.2 of Appendix A. Another test

of convergence is the number of incorrect classification that the neural network makes over its training and validation sets.

First, a small neural network is trained, with one hidden layer. As can be seen from Figure B.1, this 10–10–9 neural network does not even begin to converge. After 5000 presentations of the training set, the neural network misclassifies 71% of

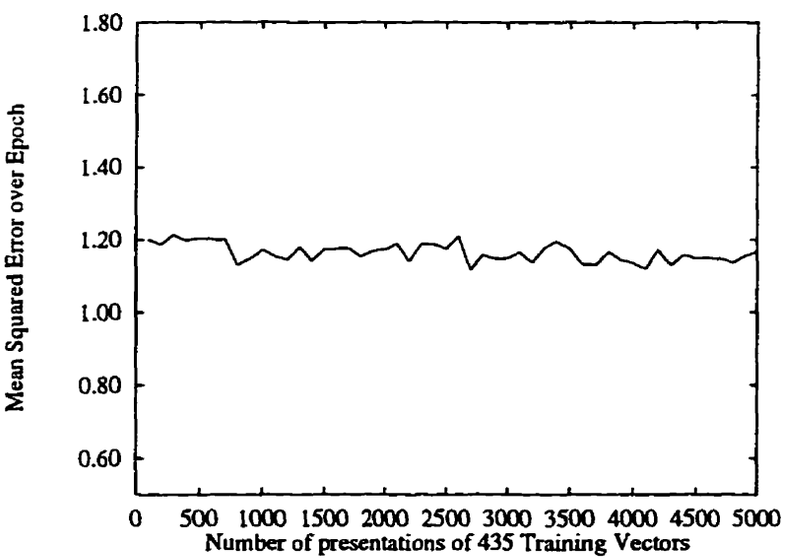


Figure B.1: Mean Squared Error for the 10–10–9 Neural Network

its training vectors. In order to increase the ability of the neural network to learn, more neurons are added to the hidden layer. One training result in this process. Figure B.2, shows the MSE of training a 10–50–9 neural network. In this case the MSE has been reduced from its initial level, but as in Figure B.1 the training bogs down. However, after 5000 presentations the error rate is reduced to 11%, with approximately the same level of MSE. This shows the reasoning of adding more

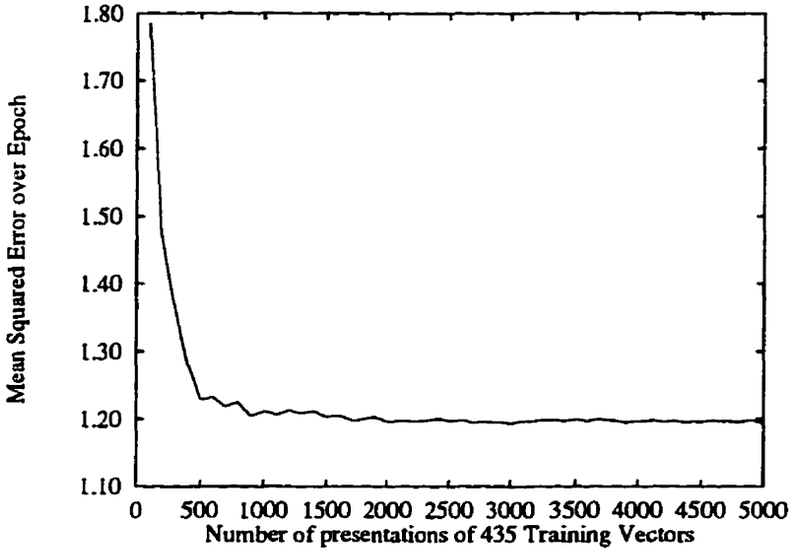


Figure B.2: Mean Squared Error for the 10–50–9 Neural Network

neurons to a neural network in order to increase its learning ability. Nonetheless, this error rate is still too high.

After further experimentation, it is discovered that an additional hidden layer is required for the neural network to converge. For example, Figure B.3 shows the training performance of the 10–10–10–9 neural network. As can be seen, the same training point is reached as with the two previous cases. As well, the classification error has increased to 39%. However, this is not surprising when compared to the 10–50–9 neural network, for it is considerably larger. Finally, if the number of neurons are increased drastically, as with the 10–100-100-9 neural network, the training results of which are shown in Figure B.4, the MSE breaks through the barrier and the neural network trains successfully, that is 0% classification error of

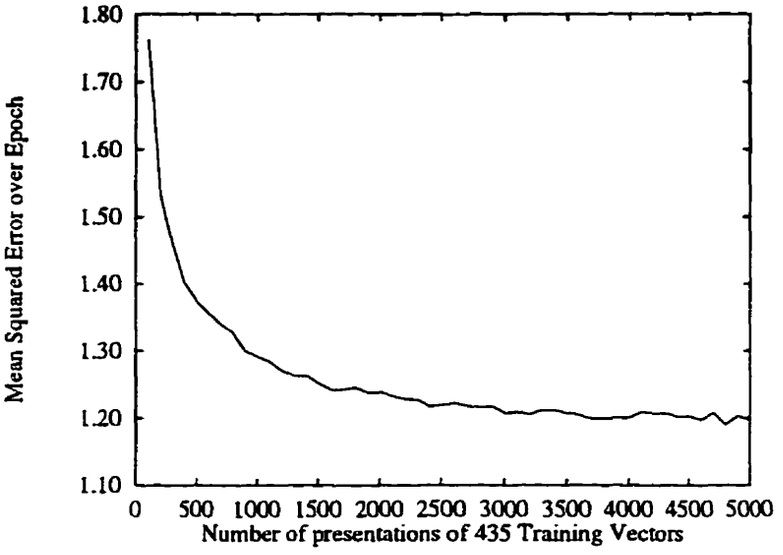


Figure B.3: Mean Squared Error for the 10–10–10–9 Neural Network

the training set. As the figure shows, this occurs in five times fewer presentations of the training vectors than in the three previous cases.

These results and others are summarized in Table B.1. It tabulates the total number synapse weights and biases in a neural network of the given size, the ratio of this number to the number of training vectors in the training set, and finally the classification error after training is thought to be complete. For the first portion of the table giving the results for ten input neural networks, there is 435 training vectors. For the second and last portions, for fifteen and twenty input neural networks, there are 2,004 and 5,996 training vectors, respectively. It is from the first third of data in the table that a heuristic is developed to help determine the number of hidden layer neurons for larger neural networks.

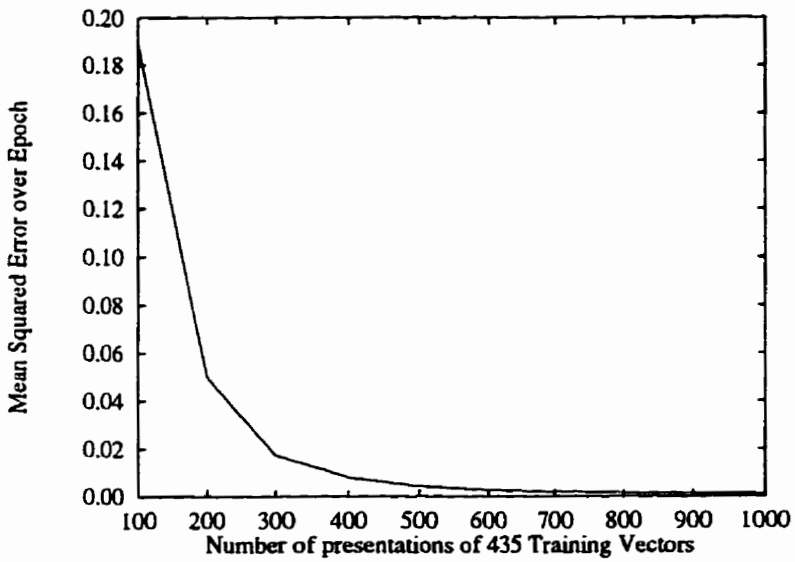


Figure B.4: Mean Squared Error for the 10–100–100–9 Neural Network

Table B.1: Training Results for Various Sizes of Neural Network

Neural Network	Number of Weights and Biases	Weights and Biases per Training Vector	Classification Error (%)
10-3-9	69	0.16	71.00
10-5-9	109	0.25	83.00
10-10-9	209	0.48	58.00
10-20-9	409	0.94	73.00
10-25-9	509	1.17	25.00
10-30-9	609	1.40	18.00
10-40-9	809	1.86	13.00
10-50-9	1009	2.32	11.00
10-80-9	1609	3.70	27.00
10-5-5-9	139	0.32	59.00
10-5-10-9	214	0.49	87.00
10-5-15-9	289	0.66	64.00
10-5-20-9	364	0.84	64.00
10-10-5-9	219	0.50	48.00
10-10-10-9	319	0.73	39.00
10-10-15-9	419	0.96	43.00
10-10-20-9	519	1.19	41.00
10-15-5-9	299	0.69	50.00
10-15-10-9	424	0.97	52.00
10-15-15-9	549	1.26	32.00

continued on next page

continued from previous page

Neural Network	Number of Weights and Biases	Weights and Biases per Training Vector	Classification Error (%)
10-15-20-9	674	1.55	18.00
10-18-15-9	627	1.44	12.00
10-20-5-9	379	0.87	39.00
10-20-10-9	529	1.22	21.00
10-20-15-9	679	1.56	12.00
10-20-20-9	829	1.91	12.00
10-22-22-9	955	2.19	2.53
10-35-35-9	1969	4.53	0.23
10-40-40-9	2449	5.63	0.23
10-50-50-9	3559	8.18	0.23
10-100-100-9	12109	27.84	0.00
15-50-50-10	3860	1.93	5.64
15-80-80-10	8570	4.28	0.00
15-100-100-10	12710	6.34	0.00
20-145-145-11	25821	4.26	0.63
20-200-200-11	46611	7.77	0.17

B.2 A Neuron Estimating Heuristic

Since much of the literature, for example [BH89, Hay94], attempts to relate the size of the neural network to the number of training vectors required to a specified

classification error level,[1] the same type of reasoning is used in the heuristic presented next. Consider the entry for the 10–35–35–9 neural network. This is the first ten input two hidden layer neural network in which training is successful. The ratio of the number of synapse weights and biases in the neural network (its size) to the number of training vectors is enumerated, simply, by counting the weights and biases and dividing by the number of training vectors. Progressing from input layer to output layer,

$$\frac{10 \cdot 35 + 35 + 35^2 + 35 + 35 \cdot 9 + 9}{435} \approx 4.53, \tag{B.1}$$

with units of "weights and biases per training vector." Now, since this size neural network converges with an acceptable classification error, if this ratio of network size to training vectors is maintained, it stands to reason that larger neural networks should also converge. Thus, the following quadratic equation is formed in estimating the number of neurons required in a fifteen input, two hidden layer and ten output neural network with 2,004 training vectors, where x represents the number of neurons sought.

$$15x + x + x^2 + x + 10x + 10 \approx 4.53 \cdot 2,004 \tag{B.2}$$

$$x^2 + 27x - 9,060.98 \approx 0 \tag{B.3}$$

[1]Unfortunately, these "rules of thumb" do not help in this particular situation, due to assumptions which do not hold.

resulting in the solutions,

$$x_1 = 82.64 \qquad \text{and} \qquad x_2 = -109.64 \tag{B.4}$$

This leads one to expect that a 15–80–80–10 neural network presented with 2,004 training vectors based on the traffic primitives of Section 2.1.3 will converge, and as show by the second portion of Table B.1, it does. Two other training results are shown in the table for fifteen input neural networks.

Using a similar strategy for estimating the number of neurons required in a twenty input, two hidden layer and eleven output neural network, with 5,996 training vectors gives

$$20x + x + x^2 + x + 11x + 11 \approx 4.53 \cdot 5,996 \tag{B.5}$$

$$x^2 + 33x - 27,129.51 \approx 0 \tag{B.6}$$

resulting in the solutions,

$$x_3 = 149.03 \qquad \text{and} \qquad x_4 = -182.03 \tag{B.7}$$

This leads one to expected that a 20–180–180–11 neural network would converge. However, from experience with the fifteen input neural networks, and considering the vast amount of time required to train a neural network of this size, the 20–145–145–11 neural network is trained. The results for this and a 20–200-200-11 neural network are stated in the third portion of Table B.1.

One may ask why the number of neurons in each hidden layer is the same. As can be seen in Table B.1, for the smaller networks some experimentation on varying this number is performed. However, the first two hidden layer neural network to converge contained the same number of neurons in each layer, and since this allows the quadratic expressions above, the convention was adopted. As will be seen in the next section, experiments in varying the number of neurons in the hidden layers of larger sized neural networks becomes extremely costly in terms of computer time.

B.3 The Training Times

The training times of only the three neural networks studied are given in Table B.2. The computer time is measured on a lightly loaded[2] Sun Sparc 20/71, operating in multi-user mode. These times are based on 3,000 presentations of the training vectors to their respective neural network. As can be seen for the two smaller neural

Table B.2: Training Times for the Three Neural Networks Studied

Neural Network	Computer Time		
	Seconds	Hours	Days
10–35–35–9	8,757	2.43	0.10
15–80–80–10	209,460	58.18	2.42
20–200–200–11	3,312,660	920.18	38.34

networks, the training times are tolerable. However, for the larger neural network,

[2]Lightly loaded should be taken to mean that the SunOS 4.1.3 command `rup` returns a load average of 1.0 for most of the training session. The training jobs run with a `nice` level of 19.

due to the exponential increase of weights with size, the training times prohibit much study of the 20 input case.

As a matter of interest, if a twenty-five input neural network is to be trained, the corresponding $W = 25$ would define 14,122 training vectors, which would require fourteen neural network outputs. Using the heuristic of the previous section, this implies that approximately 233 neurons are required. If this information is used to layout a 25–230–230–14 neural network, then the training time can be estimated to be a *minimum* of 9,480,205 seconds, or 109.72 days if the training converges within 3,000 training epochs.

Appendix C

Training Vectors and Primitive Classification Numbers

This appendix contains the training vectors for the 10–35–35–9 neural network. Tables C.1–C.4 show the training vectors for the DG, CBR, PT and CBR-RC traffic primitives, respectfully. Table C.5 summarizes the classifications performed by this primitive classifier, which is a useful reference when considering the output of the neural network. In addition, Tables C.6 and C.7 gives the primitive classification numbers for the primitive classifier based on the 15–80–80–10 and 20–200–200-11 neural networks, respectively.

The training vectors for the DG, CBR and PT traffic primitives are fairly straight forward, since they result in only one traffic classification each. In the case of DG primitives, the output desired from the neural network is the vector "000000000," using the notation introduced in Section 1.6.4.

Table C.1: Degenerate Training Vectors (11) for the 10–35-35–9 Neural Network

Primitive	Input	Output	Primitive	Input	Output
DG	1000000000	000000000	DG	0100000000	000000000
DG	0010000000	000000000	DG	0001000000	000000000
DG	0000100000	000000000	DG	0000010000	000000000
DG	0000001000	000000000	DG	0000000100	000000000
DG	0000000010	000000000	DG	0000000001	000000000
DG	0000000000	000000000			

The classification desired for the CBR primitives is "010000000."

Table C.2: Constant Bit Rate Training Vectors (25) for the 10–35–35–9 Neural Network

Primitive	Input	Output	Primitive	Input	Output
CBR	1111111111	010000000	CBR	1010101010	010000000
CBR	0101010101	010000000	CBR	1001001001	010000000
CBR	0100100100	010000000	CBR	0010010010	010000000
CBR	1000100010	010000000	CBR	0100010001	010000000
CBR	0010001000	010000000	CBR	0001000100	010000000
CBR	1000010000	010000000	CBR	0100001000	010000000
CBR	0010000100	010000000	CBR	0001000010	010000000
CBR	0000100001	010000000	CBR	1000001000	010000000
CBR	0100000100	010000000	CBR	0010000010	010000000
CBR	0001000001	010000000	CBR	1000000100	010000000
CBR	0100000010	010000000	CBR	0010000001	010000000
CBR	1000000010	010000000	CBR	0100000001	010000000
CBR	1000000001	010000000			

The classification desired for the PT primitives is "100000000."

Table C.3: Packet Train Training Vectors (156) for the 10–35–35–9 Neural Network

Primitive	Input	Output	Primitive	Input	Output
PT	1101101101	100000000	PT	1011011011	100000000
PT	0110110110	100000000	PT	1100110011	100000000
PT	1001100110	100000000	PT	0011001100	100000000
PT	0110011001	100000000	PT	1100011000	100000000
PT	1000110001	100000000	PT	0001100011	100000000
PT	0011000110	100000000	PT	0110001100	100000000
PT	1100001100	100000000	PT	1000011000	100000000
PT	0000110000	100000000	PT	0001100001	100000000
PT	0011000011	100000000	PT	0110000110	100000000
PT	1100000110	100000000	PT	1000001100	100000000
PT	0000011000	100000000	PT	0001100000	100000000
PT	0011000001	100000000	PT	0110000011	100000000
PT	1100000011	100000000	PT	1000000110	100000000
PT	0000001100	100000000	PT	0011000000	100000000
PT	0110000001	100000000	PT	1100000001	100000000
PT	1000000011	100000000	PT	0000000110	100000000
PT	0110000000	100000000	PT	1100000000	100000000
PT	0000000011	100000000	PT	1110111011	100000000
PT	1101110111	100000000	PT	1011101110	100000000
PT	0111011101	100000000	PT	1110011100	100000000
PT	1100111001	100000000	PT	1001110011	100000000
PT	0011100111	100000000	PT	0111001110	100000000
PT	1110001110	100000000	PT	1100011100	100000000
PT	1000111000	100000000	PT	0001110001	100000000
PT	0011100011	100000000	PT	0111000111	100000000
PT	1110000111	100000000	PT	1100001110	100000000
PT	1000011100	100000000	PT	0000111000	100000000
PT	0001110000	100000000	PT	0011100001	100000000

continued on next page

continued from previous page

Primitive	Input	Output	Primitive	Input	Output
PT	0111000011	100000000	PT	1110000011	100000000
PT	1100000111	100000000	PT	1000001110	100000000
PT	0000011100	100000000	PT	0011100000	100000000
PT	0111000001	100000000	PT	1110000001	100000000
PT	1000000111	100000000	PT	0000001110	100000000
PT	0111000000	100000000	PT	1110000000	100000000
PT	0000000111	100000000	PT	1111011110	100000000
PT	1110111101	100000000	PT	1101111011	100000000
PT	1011110111	100000000	PT	0111101111	100000000
PT	1111001111	100000000	PT	1110011110	100000000
PT	1100111100	100000000	PT	1001111001	100000000
PT	0011110011	100000000	PT	0111100111	100000000
PT	1111000111	100000000	PT	1110001111	100000000
PT	1100011110	100000000	PT	1000111100	100000000
PT	0001111000	100000000	PT	0011110001	100000000
PT	0111100011	100000000	PT	1111000011	100000000
PT	1100001111	100000000	PT	1000011110	100000000
PT	0000111100	100000000	PT	0011110000	100000000
PT	0111100001	100000000	PT	1111000001	100000000
PT	1000001111	100000000	PT	0000011110	100000000
PT	0111100000	100000000	PT	1111000000	100000000
PT	0000001111	100000000	PT	1111101111	100000000
PT	1111011111	100000000	PT	1110111110	100000000
PT	1101111101	100000000	PT	1011111011	100000000
PT	0111110111	100000000	PT	1111100111	100000000
PT	1110011111	100000000	PT	1100111110	100000000
PT	1001111100	100000000	PT	0011111001	100000000
PT	0111110011	100000000	PT	1111100011	100000000
PT	1100011111	100000000	PT	1000111110	100000000
PT	0001111100	100000000	PT	0011111000	100000000

continued on next page

continued from previous page

Primitive	Input	Output	Primitive	Input	Output
PT	0111110001	100000000	PT	1111100001	100000000
PT	1000011111	100000000	PT	0000111110	100000000
PT	0111110000	100000000	PT	1111100000	100000000
PT	0000011111	100000000	PT	1111110111	100000000
PT	1110111111	100000000	PT	1101111110	100000000
PT	1011111101	100000000	PT	0111111011	100000000
PT	1111110011	100000000	PT	1100111111	100000000
PT	1001111110	100000000	PT	0011111100	100000000
PT	0111111001	100000000	PT	1111110001	100000000
PT	1000111111	100000000	PT	0001111110	100000000
PT	0111111000	100000000	PT	1111110000	100000000
PT	0000111111	100000000	PT	1111111011	100000000
PT	1101111111	100000000	PT	1011111110	100000000
PT	0111111101	100000000	PT	1111111001	100000000
PT	1001111111	100000000	PT	0011111110	100000000
PT	0111111100	100000000	PT	1111111000	100000000
PT	0001111111	100000000	PT	1111111101	100000000
PT	1011111111	100000000	PT	0111111110	100000000
PT	1111111100	100000000	PT	0011111111	100000000
PT	1111111110	100000000	PT	0111111111	100000000

For the case of CBR-RC traffic primitives, instead of all the training vectors returning the same classification, a group of CBR-RC morphisms return individual classifications. For example, the first three rows of Table C.4 show six inputs to the neural network which correspond to the situation which occurs when a source transmitting at the full link rate C begins to transmit at half the link rate, $\frac{C}{2}$. This transition is represented by the RC $C \rightarrow \frac{C}{2}$ traffic primitive, and given by a neural network output of "110000001."

Table C.4: CBR Rate Change Training Vectors (243) for the 10–35–35–9 Neural Network

Primitive	Input	Output	Primitive	Input	Output
RC $C \rightarrow \frac{C}{2}$	1111111010	110000001	RC $C \rightarrow \frac{C}{2}$	1111110101	110000001
RC $C \rightarrow \frac{C}{2}$	1111101010	110000001	RC $C \rightarrow \frac{C}{2}$	1111010101	110000001
RC $C \rightarrow \frac{C}{2}$	1110101010	110000001	RC $C \rightarrow \frac{C}{2}$	1101010101	110000001
RC $C \rightarrow \frac{C}{3}$	1111110010	110000010	RC $C \rightarrow \frac{C}{3}$	1111100100	110000010
RC $C \rightarrow \frac{C}{3}$	1111001001	110000010	RC $C \rightarrow \frac{C}{3}$	1110010010	110000010
RC $C \rightarrow \frac{C}{3}$	1100100100	110000010	RC $C \rightarrow \frac{C}{3}$	1111110100	110000010
RC $C \rightarrow \frac{C}{3}$	1111101001	110000010	RC $C \rightarrow \frac{C}{3}$	1111010010	110000010
RC $C \rightarrow \frac{C}{3}$	1110100100	110000010	RC $C \rightarrow \frac{C}{3}$	1101001001	110000010
RC $C \rightarrow \frac{C}{3}$	1010010010	110000010	RC $C \rightarrow \frac{C}{4}$	1111100010	110000011
RC $C \rightarrow \frac{C}{4}$	1111000100	110000011	RC $C \rightarrow \frac{C}{4}$	1110001000	110000011
RC $C \rightarrow \frac{C}{4}$	1100010001	110000011	RC $C \rightarrow \frac{C}{4}$	1111101000	110000011
RC $C \rightarrow \frac{C}{4}$	1111010001	110000011	RC $C \rightarrow \frac{C}{4}$	1110100010	110000011
RC $C \rightarrow \frac{C}{4}$	1101000100	110000011	RC $C \rightarrow \frac{C}{4}$	1010001000	110000011
RC $C \rightarrow \frac{C}{4}$	1111001000	110000011	RC $C \rightarrow \frac{C}{4}$	1110010001	110000011
RC $C \rightarrow \frac{C}{4}$	1100100010	110000011	RC $C \rightarrow \frac{C}{4}$	1001000100	110000011
RC $C \rightarrow \frac{C}{5}$	1111000010	110000100	RC $C \rightarrow \frac{C}{5}$	1110000100	110000100
RC $C \rightarrow \frac{C}{5}$	1100001000	110000100	RC $C \rightarrow \frac{C}{5}$	1111010000	110000100
RC $C \rightarrow \frac{C}{5}$	1110100001	110000100	RC $C \rightarrow \frac{C}{5}$	1101000010	110000100
RC $C \rightarrow \frac{C}{5}$	1010000100	110000100	RC $C \rightarrow \frac{C}{5}$	1110010000	110000100
RC $C \rightarrow \frac{C}{5}$	1100100001	110000100	RC $C \rightarrow \frac{C}{5}$	1001000010	110000100
RC $C \rightarrow \frac{C}{5}$	1100010000	110000100	RC $C \rightarrow \frac{C}{5}$	1000100001	110000100
RC $C \rightarrow \frac{C}{6}$	1110000010	110000101	RC $C \rightarrow \frac{C}{6}$	1100000100	110000101
RC $C \rightarrow \frac{C}{6}$	1110100000	110000101	RC $C \rightarrow \frac{C}{6}$	1101000001	110000101
RC $C \rightarrow \frac{C}{6}$	1010000010	110000101	RC $C \rightarrow \frac{C}{6}$	1100100000	110000101
RC $C \rightarrow \frac{C}{6}$	1001000001	110000101	RC $C \rightarrow \frac{C}{6}$	1000100000	110000101
RC $C \rightarrow \frac{C}{7}$	1100000010	110000110	RC $C \rightarrow \frac{C}{7}$	1101000000	110000110
RC $C \rightarrow \frac{C}{7}$	1010000001	110000110	RC $C \rightarrow \frac{C}{7}$	1001000000	110000110
RC $C \rightarrow \frac{C}{8}$	1010000000	110000111	RC $\frac{C}{2} \rightarrow C$	1010101011	110001010
RC $\frac{C}{2} \rightarrow C$	0101010111	110001010	RC $\frac{C}{2} \rightarrow C$	1010101111	110001010

continued on next page

continued from previous page

Primitive	Input	Output	Primitive	Input	Output
RC $\frac{C}{2} \rightarrow C$	0101011111	1100010 10	RC $\frac{C}{2} \rightarrow C$	1010111111	1100010 10
RC $\frac{C}{2} \rightarrow C$	0101111111	1100010 10	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0101010100	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	1010101001	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0101010010	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	1010100100	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0101001001	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	1010101000	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0101010001	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	1010100010	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0101000100	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	1010001001	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0100010010	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	1000100100	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0001001001	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0101010110	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	1010101100	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0101011001	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	1010110010	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0101100100	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	1011001001	1100010 11
RC $\frac{C}{2} \rightarrow \frac{C}{3}$	0110010010	1100010 11	RC $\frac{C}{2} \rightarrow \frac{C}{4}$	0101001000	1100011 00
RC $\frac{C}{2} \rightarrow \frac{C}{4}$	1010010001	1100011 00	RC $\frac{C}{2} \rightarrow \frac{C}{4}$	0100100010	1100011 00
RC $\frac{C}{2} \rightarrow \frac{C}{4}$	0101010000	1100011 00	RC $\frac{C}{2} \rightarrow \frac{C}{4}$	1010100001	1100011 00
RC $\frac{C}{2} \rightarrow \frac{C}{4}$	0101000010	1100011 00	RC $\frac{C}{2} \rightarrow \frac{C}{4}$	1000010001	1100011 00
RC $\frac{C}{2} \rightarrow \frac{C}{4}$	0000100010	1100011 00	RC $\frac{C}{2} \rightarrow \frac{C}{4}$	0101011000	1100011 00
RC $\frac{C}{2} \rightarrow \frac{C}{4}$	1010110001	1100011 00	RC $\frac{C}{2} \rightarrow \frac{C}{4}$	0101100010	1100011 00
RC $\frac{C}{2} \rightarrow \frac{C}{4}$	1011000100	1100011 00	RC $\frac{C}{2} \rightarrow \frac{C}{4}$	0110001000	1100011 00
RC $\frac{C}{2} \rightarrow \frac{C}{5}$	1010010000	1100011 01	RC $\frac{C}{2} \rightarrow \frac{C}{5}$	0100100001	1100011 01
RC $\frac{C}{2} \rightarrow \frac{C}{5}$	0100010000	1100011 01	RC $\frac{C}{2} \rightarrow \frac{C}{5}$	1010100000	1100011 01
RC $\frac{C}{2} \rightarrow \frac{C}{5}$	0101000001	1100011 01	RC $\frac{C}{2} \rightarrow \frac{C}{5}$	1010110000	1100011 01
RC $\frac{C}{2} \rightarrow \frac{C}{5}$	0101100001	1100011 01	RC $\frac{C}{2} \rightarrow \frac{C}{5}$	1011000010	1100011 01
RC $\frac{C}{2} \rightarrow \frac{C}{5}$	0110000100	1100011 01	RC $\frac{C}{2} \rightarrow \frac{C}{6}$	0100100000	1100011 10
RC $\frac{C}{2} \rightarrow \frac{C}{6}$	0101100000	1100011 10	RC $\frac{C}{2} \rightarrow \frac{C}{6}$	1011000001	1100011 10
RC $\frac{C}{2} \rightarrow \frac{C}{6}$	0110000010	1100011 10	RC $\frac{C}{2} \rightarrow \frac{C}{7}$	0101000000	1100011 11
RC $\frac{C}{2} \rightarrow \frac{C}{7}$	1011000000	1100011 11	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0010010011	1100100 11
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0100100110	1100100 11	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	1001001101	1100100 11
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0010011010	1100100 11	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0100110101	1100100 11
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	1001101010	1100100 11	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0011010101	1100100 11
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0110101010	1100100 11	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0100100101	1100100 11

continued on next page

continued from previous page

Primitive	Input	Output	Primitive	Input	Output
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	1001001010	110010011	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0010010101	110010011
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0100101010	110010011	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	1001010101	110010011
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0010101010	110010011	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	1001001000	110010011
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0010010001	110010011	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	1001000101	110010011
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0010001010	110010011	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0100010101	110010011
RC $\frac{C}{3} \rightarrow \frac{C}{2}$	1000101010	110010011	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	0001010101	110010011
RC $\frac{C}{3} \rightarrow C$	0100100111	110010100	RC $\frac{C}{3} \rightarrow C$	1001001111	110010100
RC $\frac{C}{3} \rightarrow C$	0010011111	110010100	RC $\frac{C}{3} \rightarrow C$	0100111111	110010100
RC $\frac{C}{3} \rightarrow C$	1001001011	110010100	RC $\frac{C}{3} \rightarrow C$	0010010111	110010100
RC $\frac{C}{3} \rightarrow C$	0100101111	110010100	RC $\frac{C}{3} \rightarrow C$	1001011111	110010100
RC $\frac{C}{3} \rightarrow C$	0010111111	110010100	RC $\frac{C}{3} \rightarrow \frac{C}{4}$	1001001100	110010101
RC $\frac{C}{3} \rightarrow \frac{C}{4}$	0010011000	110010101	RC $\frac{C}{3} \rightarrow \frac{C}{4}$	0100110001	110010101
RC $\frac{C}{3} \rightarrow \frac{C}{4}$	1001100010	110010101	RC $\frac{C}{3} \rightarrow \frac{C}{4}$	0011000100	110010101
RC $\frac{C}{3} \rightarrow \frac{C}{4}$	0010010100	110010101	RC $\frac{C}{3} \rightarrow \frac{C}{4}$	0100101000	110010101
RC $\frac{C}{3} \rightarrow \frac{C}{4}$	1001010001	110010101	RC $\frac{C}{3} \rightarrow \frac{C}{4}$	0010100010	110010101
RC $\frac{C}{3} \rightarrow \frac{C}{4}$	0010010000	110010101	RC $\frac{C}{3} \rightarrow \frac{C}{4}$	0000010001	110010101
RC $\frac{C}{3} \rightarrow \frac{C}{5}$	0100110000	110010110	RC $\frac{C}{3} \rightarrow \frac{C}{5}$	1001100001	110010110
RC $\frac{C}{3} \rightarrow \frac{C}{5}$	0011000010	110010110	RC $\frac{C}{3} \rightarrow \frac{C}{5}$	1001010000	110010110
RC $\frac{C}{3} \rightarrow \frac{C}{5}$	0010100001	110010110	RC $\frac{C}{3} \rightarrow \frac{C}{6}$	1001100000	110010111
RC $\frac{C}{3} \rightarrow \frac{C}{6}$	0010100000	110010111	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0001000101	110011100
RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0100010100	110011100	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	1000101001	110011100
RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0001010010	110011100	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0010100100	110011100
RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0010001001	110011100	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	1000100011	110011100
RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0001000110	110011100	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0010001100	110011100
RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0100011001	110011100	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	1000110010	110011100
RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0001100100	110011100	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0011001001	110011100
RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0100001001	110011100	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	1000010010	110011100
RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0000100100	110011100	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	1000001001	110011100
RC $\frac{C}{4} \rightarrow \frac{C}{3}$	0000010010	110011100	RC $\frac{C}{4} \rightarrow \frac{C}{2}$	1000100101	110011101
RC $\frac{C}{4} \rightarrow \frac{C}{2}$	0001001010	110011101	RC $\frac{C}{4} \rightarrow \frac{C}{2}$	0010001101	110011101

continued on next page

continued from previous page

Primitive	Input	Output	Primitive	Input	Output
RC $\frac{C}{4} \rightarrow \frac{C}{2}$	0100011010	1100111101	RC $\frac{C}{4} \rightarrow \frac{C}{2}$	1000110101	1100111101
RC $\frac{C}{4} \rightarrow \frac{C}{2}$	0001101010	1100111101	RC $\frac{C}{4} \rightarrow \frac{C}{2}$	0010000101	1100111101
RC $\frac{C}{4} \rightarrow \frac{C}{2}$	0100001010	1100111101	RC $\frac{C}{4} \rightarrow \frac{C}{2}$	1000010101	1100111101
RC $\frac{C}{4} \rightarrow \frac{C}{2}$	0000101010	1100111101	RC $\frac{C}{4} \rightarrow C$	0010001011	1100111110
RC $\frac{C}{4} \rightarrow C$	0100010111	1100111110	RC $\frac{C}{4} \rightarrow C$	1000101111	1100111110
RC $\frac{C}{4} \rightarrow C$	0001011111	1100111110	RC $\frac{C}{4} \rightarrow C$	0001000111	1100111110
RC $\frac{C}{4} \rightarrow C$	0010001111	1100111110	RC $\frac{C}{4} \rightarrow C$	0100011111	1100111110
RC $\frac{C}{4} \rightarrow C$	0100010011	1100111110	RC $\frac{C}{4} \rightarrow C$	1000100111	1100111110
RC $\frac{C}{4} \rightarrow C$	0001001111	1100111110	RC $\frac{C}{4} \rightarrow \frac{C}{5}$	1000101000	1100111111
RC $\frac{C}{4} \rightarrow \frac{C}{5}$	0001010000	1100111111	RC $\frac{C}{4} \rightarrow \frac{C}{5}$	0001001000	1100111111
RC $\frac{C}{4} \rightarrow \frac{C}{5}$	0100011000	1100111111	RC $\frac{C}{4} \rightarrow \frac{C}{5}$	1000110000	1100111111
RC $\frac{C}{5} \rightarrow \frac{C}{4}$	1000010100	1101001101	RC $\frac{C}{5} \rightarrow \frac{C}{4}$	0000101000	1101001101
RC $\frac{C}{5} \rightarrow \frac{C}{4}$	0001010001	1101001101	RC $\frac{C}{5} \rightarrow \frac{C}{4}$	0001000011	1101001101
RC $\frac{C}{5} \rightarrow \frac{C}{4}$	0010000110	1101001101	RC $\frac{C}{5} \rightarrow \frac{C}{4}$	0100001100	1101001101
RC $\frac{C}{5} \rightarrow \frac{C}{4}$	0000110001	1101001101	RC $\frac{C}{5} \rightarrow \frac{C}{4}$	0001100010	1101001101
RC $\frac{C}{5} \rightarrow \frac{C}{3}$	0000001001	1101001110	RC $\frac{C}{5} \rightarrow \frac{C}{3}$	0000101001	1101001110
RC $\frac{C}{5} \rightarrow \frac{C}{3}$	1000011001	1101001110	RC $\frac{C}{5} \rightarrow \frac{C}{3}$	0000110010	1101001110
RC $\frac{C}{5} \rightarrow \frac{C}{2}$	0100000101	1101001111	RC $\frac{C}{5} \rightarrow \frac{C}{2}$	1000001010	1101001111
RC $\frac{C}{5} \rightarrow \frac{C}{2}$	0000010101	1101001111	RC $\frac{C}{5} \rightarrow \frac{C}{2}$	0000100101	1101001111
RC $\frac{C}{5} \rightarrow \frac{C}{2}$	0100001101	1101001111	RC $\frac{C}{5} \rightarrow \frac{C}{2}$	1000011010	1101001111
RC $\frac{C}{5} \rightarrow \frac{C}{2}$	0000110101	1101001111	RC $\frac{C}{5} \rightarrow C$	0010000111	1101010000
RC $\frac{C}{5} \rightarrow C$	0100001111	1101010000	RC $\frac{C}{5} \rightarrow C$	0000100011	1101010000
RC $\frac{C}{5} \rightarrow C$	1000010011	1101010000	RC $\frac{C}{5} \rightarrow C$	0000100111	1101010000
RC $\frac{C}{5} \rightarrow C$	0100001011	1101010000	RC $\frac{C}{5} \rightarrow C$	1000010111	1101010000
RC $\frac{C}{5} \rightarrow C$	0000101111	1101010000	RC $\frac{C}{6} \rightarrow \frac{C}{5}$	0000010100	1101011100
RC $\frac{C}{6} \rightarrow \frac{C}{5}$	0010000011	1101011100	RC $\frac{C}{6} \rightarrow \frac{C}{5}$	0100000110	1101011100
RC $\frac{C}{6} \rightarrow \frac{C}{3}$	0000011001	1101100001	RC $\frac{C}{6} \rightarrow \frac{C}{2}$	1000001101	1101100011
RC $\frac{C}{6} \rightarrow \frac{C}{2}$	0000011010	1101100011	RC $\frac{C}{6} \rightarrow C$	0000010011	1101100100
RC $\frac{C}{6} \rightarrow C$	1000001011	1101100100	RC $\frac{C}{6} \rightarrow C$	0000010111	1101100100
RC $\frac{C}{6} \rightarrow C$	0100000111	1101100100	RC $\frac{C}{7} \rightarrow \frac{C}{6}$	1000000101	1101101111

continued on next page

continued from previous page

Primitive	Input	Output	Primitive	Input	Output
RC $\frac{C}{7} \rightarrow \frac{C}{6}$	0000001010	110110111	RC $\frac{C}{7} \rightarrow \frac{C}{6}$	0100000011	110110111
RC $\frac{C}{7} \rightarrow \frac{C}{2}$	0000001101	110111011	RC $\frac{C}{7} \rightarrow C$	0000001011	110111100
RC $\frac{C}{8} \rightarrow \frac{C}{7}$	0000000101	111000000			

The output of the primitive classifier is a binary coded classification, and hence is not in a very human-readable format. Thus, each classification is given a number, as summarized in Table C.5 for a classifier based on the 10–35–35–9 neural network. This table is useful when observing the output graphs of the primitive classifier.

Table C.5: Summary of Primitive Classifications, including Classification Number for the 10–35–35–9 Neural Network

Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive
1	DG	2	CBR	3	PT
4	RC $C \rightarrow \frac{C}{2}$	5	RC $C \rightarrow \frac{C}{3}$	6	RC $C \rightarrow \frac{C}{4}$
7	RC $C \rightarrow \frac{C}{5}$	8	RC $C \rightarrow \frac{C}{6}$	9	RC $C \rightarrow \frac{C}{7}$
10	RC $C \rightarrow \frac{C}{8}$	11	RC $\frac{C}{2} \rightarrow C$	12	RC $\frac{C}{2} \rightarrow \frac{C}{3}$
13	RC $\frac{C}{2} \rightarrow \frac{C}{4}$	14	RC $\frac{C}{2} \rightarrow \frac{C}{5}$	15	RC $\frac{C}{2} \rightarrow \frac{C}{6}$
16	RC $\frac{C}{2} \rightarrow \frac{C}{7}$	17	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	18	RC $\frac{C}{3} \rightarrow C$
19	RC $\frac{C}{3} \rightarrow \frac{C}{4}$	20	RC $\frac{C}{3} \rightarrow \frac{C}{5}$	21	RC $\frac{C}{3} \rightarrow \frac{C}{6}$
22	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	23	RC $\frac{C}{4} \rightarrow \frac{C}{2}$	24	RC $\frac{C}{4} \rightarrow C$
25	RC $\frac{C}{4} \rightarrow \frac{C}{5}$	26	RC $\frac{C}{5} \rightarrow \frac{C}{4}$	27	RC $\frac{C}{5} \rightarrow \frac{C}{3}$
28	RC $\frac{C}{5} \rightarrow \frac{C}{2}$	29	RC $\frac{C}{5} \rightarrow C$	30	RC $\frac{C}{6} \rightarrow \frac{C}{5}$
31	RC $\frac{C}{6} \rightarrow \frac{C}{3}$	32	RC $\frac{C}{6} \rightarrow \frac{C}{2}$	33	RC $\frac{C}{6} \rightarrow C$
34	RC $\frac{C}{7} \rightarrow \frac{C}{6}$	35	RC $\frac{C}{7} \rightarrow \frac{C}{2}$	36	RC $\frac{C}{7} \rightarrow C$
37	RC $\frac{C}{8} \rightarrow \frac{C}{7}$				

The traffic primitive classifications are enumerated in Table C.6 for the 15–80–80–10 neural network based primitive classifier. This table is useful when observing the output graphs of the classifier.

Table C.6: Summary of Primitive Classifications, including Classification Number for the 15–80–80–10 Neural Network

Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive
1	DG	2	CBR	3	PT
4	RC $C \rightarrow \frac{C}{2}$	5	RC $C \rightarrow \frac{C}{3}$	6	RC $C \rightarrow \frac{C}{4}$
7	RC $C \rightarrow \frac{C}{5}$	8	RC $C \rightarrow \frac{C}{6}$	9	RC $C \rightarrow \frac{C}{7}$
10	RC $C \rightarrow \frac{C}{8}$	11	RC $C \rightarrow \frac{C}{9}$	12	RC $C \rightarrow \frac{C}{10}$
13	RC $C \rightarrow \frac{C}{11}$	14	RC $C \rightarrow \frac{C}{12}$	15	RC $C \rightarrow \frac{C}{13}$
16	RC $\frac{C}{2} \rightarrow C$	17	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	18	RC $\frac{C}{2} \rightarrow \frac{C}{4}$
19	RC $\frac{C}{2} \rightarrow \frac{C}{5}$	20	RC $\frac{C}{2} \rightarrow \frac{C}{6}$	21	RC $\frac{C}{2} \rightarrow \frac{C}{7}$
22	RC $\frac{C}{2} \rightarrow \frac{C}{8}$	23	RC $\frac{C}{2} \rightarrow \frac{C}{9}$	24	RC $\frac{C}{2} \rightarrow \frac{C}{10}$
25	RC $\frac{C}{2} \rightarrow \frac{C}{11}$	26	RC $\frac{C}{2} \rightarrow \frac{C}{12}$	27	RC $\frac{C}{3} \rightarrow \frac{C}{2}$
28	RC $\frac{C}{3} \rightarrow C$	29	RC $\frac{C}{3} \rightarrow \frac{C}{4}$	30	RC $\frac{C}{3} \rightarrow \frac{C}{5}$
31	RC $\frac{C}{3} \rightarrow \frac{C}{6}$	32	RC $\frac{C}{3} \rightarrow \frac{C}{7}$	33	RC $\frac{C}{3} \rightarrow \frac{C}{8}$
34	RC $\frac{C}{3} \rightarrow \frac{C}{9}$	35	RC $\frac{C}{3} \rightarrow \frac{C}{10}$	36	RC $\frac{C}{3} \rightarrow \frac{C}{11}$
37	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	38	RC $\frac{C}{4} \rightarrow \frac{C}{2}$	39	RC $\frac{C}{4} \rightarrow C$
40	RC $\frac{C}{4} \rightarrow \frac{C}{5}$	41	RC $\frac{C}{4} \rightarrow \frac{C}{6}$	42	RC $\frac{C}{4} \rightarrow \frac{C}{7}$
43	RC $\frac{C}{4} \rightarrow \frac{C}{8}$	44	RC $\frac{C}{4} \rightarrow \frac{C}{9}$	45	RC $\frac{C}{4} \rightarrow \frac{C}{10}$
46	RC $\frac{C}{5} \rightarrow \frac{C}{4}$	47	RC $\frac{C}{5} \rightarrow \frac{C}{3}$	48	RC $\frac{C}{5} \rightarrow \frac{C}{2}$
49	RC $\frac{C}{5} \rightarrow C$	50	RC $\frac{C}{5} \rightarrow \frac{C}{6}$	51	RC $\frac{C}{5} \rightarrow \frac{C}{7}$
52	RC $\frac{C}{5} \rightarrow \frac{C}{8}$	53	RC $\frac{C}{5} \rightarrow \frac{C}{9}$	54	RC $\frac{C}{6} \rightarrow \frac{C}{5}$
55	RC $\frac{C}{6} \rightarrow \frac{C}{4}$	56	RC $\frac{C}{6} \rightarrow \frac{C}{3}$	57	RC $\frac{C}{6} \rightarrow \frac{C}{2}$
58	RC $\frac{C}{6} \rightarrow C$	59	RC $\frac{C}{6} \rightarrow \frac{C}{7}$	60	RC $\frac{C}{6} \rightarrow \frac{C}{8}$
61	RC $\frac{C}{7} \rightarrow \frac{C}{6}$	62	RC $\frac{C}{7} \rightarrow \frac{C}{5}$	63	RC $\frac{C}{7} \rightarrow \frac{C}{4}$
64	RC $\frac{C}{7} \rightarrow \frac{C}{3}$	65	RC $\frac{C}{7} \rightarrow \frac{C}{2}$	66	RC $\frac{C}{7} \rightarrow C$
67	RC $\frac{C}{7} \rightarrow \frac{C}{8}$	68	RC $\frac{C}{8} \rightarrow \frac{C}{7}$	69	RC $\frac{C}{8} \rightarrow \frac{C}{6}$
70	RC $\frac{C}{8} \rightarrow \frac{C}{5}$	71	RC $\frac{C}{8} \rightarrow \frac{C}{4}$	72	RC $\frac{C}{8} \rightarrow \frac{C}{3}$
73	RC $\frac{C}{8} \rightarrow \frac{C}{2}$	74	RC $\frac{C}{8} \rightarrow C$	75	RC $\frac{C}{9} \rightarrow \frac{C}{8}$
76	RC $\frac{C}{9} \rightarrow \frac{C}{5}$	77	RC $\frac{C}{9} \rightarrow \frac{C}{4}$	78	RC $\frac{C}{9} \rightarrow \frac{C}{3}$
79	RC $\frac{C}{9} \rightarrow \frac{C}{2}$	80	RC $\frac{C}{9} \rightarrow C$	81	RC $\frac{C}{10} \rightarrow \frac{C}{9}$
82	RC $\frac{C}{10} \rightarrow \frac{C}{4}$	83	RC $\frac{C}{10} \rightarrow \frac{C}{3}$	84	RC $\frac{C}{10} \rightarrow \frac{C}{2}$

continued on next page

continued from previous page

Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive
85	RC $\frac{C}{10} \rightarrow C$	86	RC $\frac{C}{11} \rightarrow \frac{C}{10}$	87	RC $\frac{C}{11} \rightarrow \frac{C}{3}$
88	RC $\frac{C}{11} \rightarrow \frac{C}{2}$	89	RC $\frac{C}{11} \rightarrow C$	90	RC $\frac{C}{12} \rightarrow \frac{C}{11}$
91	RC $\frac{C}{12} \rightarrow \frac{C}{2}$	92	RC $\frac{C}{12} \rightarrow C$	93	RC $\frac{C}{13} \rightarrow \frac{C}{12}$

The traffic primitive classifications are enumerated in Table C.7 for the 20–200–200–11 neural network based primitive classifier. This table is useful when observing the output graphs of the classifier.

Table C.7: Summary of Primitive Classifications, including Classification Number for the 20–200–200–11 Neural Network

Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive
1	DG	2	CBR	3	PT
4	RC $C \rightarrow \frac{C}{2}$	5	RC $C \rightarrow \frac{C}{3}$	6	RC $C \rightarrow \frac{C}{4}$
7	RC $C \rightarrow \frac{C}{5}$	8	RC $C \rightarrow \frac{C}{6}$	9	RC $C \rightarrow \frac{C}{7}$
10	RC $C \rightarrow \frac{C}{8}$	11	RC $C \rightarrow \frac{C}{9}$	12	RC $C \rightarrow \frac{C}{10}$
13	RC $C \rightarrow \frac{C}{11}$	14	RC $C \rightarrow \frac{C}{12}$	15	RC $C \rightarrow \frac{C}{13}$
16	RC $C \rightarrow \frac{C}{14}$	17	RC $C \rightarrow \frac{C}{15}$	18	RC $C \rightarrow \frac{C}{16}$
19	RC $C \rightarrow \frac{C}{17}$	20	RC $C \rightarrow \frac{C}{18}$	21	RC $\frac{C}{2} \rightarrow C$
22	RC $\frac{C}{2} \rightarrow \frac{C}{3}$	23	RC $\frac{C}{2} \rightarrow \frac{C}{4}$	24	RC $\frac{C}{2} \rightarrow \frac{C}{5}$
25	RC $\frac{C}{2} \rightarrow \frac{C}{6}$	26	RC $\frac{C}{2} \rightarrow \frac{C}{7}$	27	RC $\frac{C}{2} \rightarrow \frac{C}{8}$
28	RC $\frac{C}{2} \rightarrow \frac{C}{9}$	29	RC $\frac{C}{2} \rightarrow \frac{C}{10}$	30	RC $\frac{C}{2} \rightarrow \frac{C}{11}$
31	RC $\frac{C}{2} \rightarrow \frac{C}{12}$	32	RC $\frac{C}{2} \rightarrow \frac{C}{13}$	33	RC $\frac{C}{2} \rightarrow \frac{C}{14}$
34	RC $\frac{C}{2} \rightarrow \frac{C}{15}$	35	RC $\frac{C}{2} \rightarrow \frac{C}{16}$	36	RC $\frac{C}{2} \rightarrow \frac{C}{17}$
37	RC $\frac{C}{3} \rightarrow \frac{C}{2}$	38	RC $\frac{C}{3} \rightarrow C$	39	RC $\frac{C}{3} \rightarrow \frac{C}{4}$
40	RC $\frac{C}{3} \rightarrow \frac{C}{5}$	41	RC $\frac{C}{3} \rightarrow \frac{C}{6}$	42	RC $\frac{C}{3} \rightarrow \frac{C}{7}$
43	RC $\frac{C}{3} \rightarrow \frac{C}{8}$	44	RC $\frac{C}{3} \rightarrow \frac{C}{9}$	45	RC $\frac{C}{3} \rightarrow \frac{C}{10}$
46	RC $\frac{C}{3} \rightarrow \frac{C}{11}$	47	RC $\frac{C}{3} \rightarrow \frac{C}{12}$	48	RC $\frac{C}{3} \rightarrow \frac{C}{13}$
49	RC $\frac{C}{3} \rightarrow \frac{C}{14}$	50	RC $\frac{C}{3} \rightarrow \frac{C}{15}$	51	RC $\frac{C}{3} \rightarrow \frac{C}{16}$
52	RC $\frac{C}{4} \rightarrow \frac{C}{3}$	53	RC $\frac{C}{4} \rightarrow \frac{C}{2}$	54	RC $\frac{C}{4} \rightarrow C$

continued on next page

continued from previous page

Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive
55	RC $\frac{C}{4} \rightarrow \frac{C}{5}$	56	RC $\frac{C}{4} \rightarrow \frac{C}{6}$	57	RC $\frac{C}{4} \rightarrow \frac{C}{7}$
58	RC $\frac{C}{4} \rightarrow \frac{C}{8}$	59	RC $\frac{C}{4} \rightarrow \frac{C}{9}$	60	RC $\frac{C}{4} \rightarrow \frac{C}{10}$
61	RC $\frac{C}{4} \rightarrow \frac{C}{11}$	62	RC $\frac{C}{4} \rightarrow \frac{C}{12}$	63	RC $\frac{C}{4} \rightarrow \frac{C}{13}$
64	RC $\frac{C}{4} \rightarrow \frac{C}{14}$	65	RC $\frac{C}{4} \rightarrow \frac{C}{15}$	66	RC $\frac{C}{5} \rightarrow \frac{C}{4}$
67	RC $\frac{C}{5} \rightarrow \frac{C}{3}$	68	RC $\frac{C}{5} \rightarrow \frac{C}{2}$	69	RC $\frac{C}{5} \rightarrow C$
70	RC $\frac{C}{5} \rightarrow \frac{C}{6}$	71	RC $\frac{C}{5} \rightarrow \frac{C}{7}$	72	RC $\frac{C}{5} \rightarrow \frac{C}{8}$
73	RC $\frac{C}{5} \rightarrow \frac{C}{9}$	74	RC $\frac{C}{5} \rightarrow \frac{C}{10}$	75	RC $\frac{C}{5} \rightarrow \frac{C}{11}$
76	RC $\frac{C}{5} \rightarrow \frac{C}{12}$	77	RC $\frac{C}{5} \rightarrow \frac{C}{13}$	78	RC $\frac{C}{5} \rightarrow \frac{C}{14}$
79	RC $\frac{C}{6} \rightarrow \frac{C}{5}$	80	RC $\frac{C}{6} \rightarrow \frac{C}{4}$	81	RC $\frac{C}{6} \rightarrow \frac{C}{3}$
82	RC $\frac{C}{6} \rightarrow \frac{C}{2}$	83	RC $\frac{C}{6} \rightarrow C$	84	RC $\frac{C}{6} \rightarrow \frac{C}{7}$
85	RC $\frac{C}{6} \rightarrow \frac{C}{8}$	86	RC $\frac{C}{6} \rightarrow \frac{C}{9}$	87	RC $\frac{C}{6} \rightarrow \frac{C}{10}$
88	RC $\frac{C}{6} \rightarrow \frac{C}{11}$	89	RC $\frac{C}{6} \rightarrow \frac{C}{12}$	90	RC $\frac{C}{6} \rightarrow \frac{C}{13}$
91	RC $\frac{C}{7} \rightarrow \frac{C}{6}$	92	RC $\frac{C}{7} \rightarrow \frac{C}{5}$	93	RC $\frac{C}{7} \rightarrow \frac{C}{4}$
94	RC $\frac{C}{7} \rightarrow \frac{C}{3}$	95	RC $\frac{C}{7} \rightarrow \frac{C}{2}$	96	RC $\frac{C}{7} \rightarrow C$
97	RC $\frac{C}{7} \rightarrow \frac{C}{8}$	98	RC $\frac{C}{7} \rightarrow \frac{C}{9}$	99	RC $\frac{C}{7} \rightarrow \frac{C}{10}$
100	RC $\frac{C}{7} \rightarrow \frac{C}{11}$	101	RC $\frac{C}{7} \rightarrow \frac{C}{12}$	102	RC $\frac{C}{8} \rightarrow \frac{C}{7}$
103	RC $\frac{C}{8} \rightarrow \frac{C}{6}$	104	RC $\frac{C}{8} \rightarrow \frac{C}{5}$	105	RC $\frac{C}{8} \rightarrow \frac{C}{4}$
106	RC $\frac{C}{8} \rightarrow \frac{C}{3}$	107	RC $\frac{C}{8} \rightarrow \frac{C}{2}$	108	RC $\frac{C}{8} \rightarrow C$
109	RC $\frac{C}{8} \rightarrow \frac{C}{9}$	110	RC $\frac{C}{8} \rightarrow \frac{C}{10}$	111	RC $\frac{C}{8} \rightarrow \frac{C}{11}$
112	RC $\frac{C}{9} \rightarrow \frac{C}{8}$	113	RC $\frac{C}{9} \rightarrow \frac{C}{7}$	114	RC $\frac{C}{9} \rightarrow \frac{C}{6}$
115	RC $\frac{C}{9} \rightarrow \frac{C}{5}$	116	RC $\frac{C}{9} \rightarrow \frac{C}{4}$	117	RC $\frac{C}{9} \rightarrow \frac{C}{3}$
118	RC $\frac{C}{9} \rightarrow \frac{C}{2}$	119	RC $\frac{C}{9} \rightarrow C$	120	RC $\frac{C}{9} \rightarrow \frac{C}{10}$
121	RC $\frac{C}{10} \rightarrow \frac{C}{9}$	122	RC $\frac{C}{10} \rightarrow \frac{C}{8}$	123	RC $\frac{C}{10} \rightarrow \frac{C}{7}$
124	RC $\frac{C}{10} \rightarrow \frac{C}{6}$	125	RC $\frac{C}{10} \rightarrow \frac{C}{5}$	126	RC $\frac{C}{10} \rightarrow \frac{C}{4}$
127	RC $\frac{C}{10} \rightarrow \frac{C}{3}$	128	RC $\frac{C}{10} \rightarrow \frac{C}{2}$	129	RC $\frac{C}{10} \rightarrow C$
130	RC $\frac{C}{11} \rightarrow \frac{C}{10}$	131	RC $\frac{C}{11} \rightarrow \frac{C}{8}$	132	RC $\frac{C}{11} \rightarrow \frac{C}{7}$
133	RC $\frac{C}{11} \rightarrow \frac{C}{6}$	134	RC $\frac{C}{11} \rightarrow \frac{C}{5}$	135	RC $\frac{C}{11} \rightarrow \frac{C}{4}$
136	RC $\frac{C}{11} \rightarrow \frac{C}{3}$	137	RC $\frac{C}{11} \rightarrow \frac{C}{2}$	138	RC $\frac{C}{11} \rightarrow C$
139	RC $\frac{C}{12} \rightarrow \frac{C}{11}$	140	RC $\frac{C}{12} \rightarrow \frac{C}{7}$	141	RC $\frac{C}{12} \rightarrow \frac{C}{6}$

continued on next page

continued from previous page

Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive	Classification Number	Corresponding Primitive
142	RC $\frac{C}{12} \rightarrow \frac{C}{5}$	143	RC $\frac{C}{12} \rightarrow \frac{C}{4}$	144	RC $\frac{C}{12} \rightarrow \frac{C}{3}$
145	RC $\frac{C}{12} \rightarrow \frac{C}{2}$	146	RC $\frac{C}{12} \rightarrow C$	147	RC $\frac{C}{13} \rightarrow \frac{C}{12}$
148	RC $\frac{C}{13} \rightarrow \frac{C}{6}$	149	RC $\frac{C}{13} \rightarrow \frac{C}{5}$	150	RC $\frac{C}{13} \rightarrow \frac{C}{4}$
151	RC $\frac{C}{13} \rightarrow \frac{C}{3}$	152	RC $\frac{C}{13} \rightarrow \frac{C}{2}$	153	RC $\frac{C}{13} \rightarrow C$
154	RC $\frac{C}{14} \rightarrow \frac{C}{13}$	155	RC $\frac{C}{14} \rightarrow \frac{C}{5}$	156	RC $\frac{C}{14} \rightarrow \frac{C}{4}$
157	RC $\frac{C}{14} \rightarrow \frac{C}{3}$	158	RC $\frac{C}{14} \rightarrow \frac{C}{2}$	159	RC $\frac{C}{14} \rightarrow C$
160	RC $\frac{C}{15} \rightarrow \frac{C}{14}$	161	RC $\frac{C}{15} \rightarrow \frac{C}{4}$	162	RC $\frac{C}{15} \rightarrow \frac{C}{3}$
163	RC $\frac{C}{15} \rightarrow \frac{C}{2}$	164	RC $\frac{C}{15} \rightarrow C$	165	RC $\frac{C}{16} \rightarrow \frac{C}{15}$
166	RC $\frac{C}{16} \rightarrow \frac{C}{3}$	167	RC $\frac{C}{16} \rightarrow \frac{C}{2}$	168	RC $\frac{C}{16} \rightarrow C$
169	RC $\frac{C}{17} \rightarrow \frac{C}{16}$	170	RC $\frac{C}{17} \rightarrow \frac{C}{2}$	171	RC $\frac{C}{17} \rightarrow C$
172	RC $\frac{C}{18} \rightarrow \frac{C}{17}$				

Bibliography

[BAF+88] Peter Bocker, Gerhard Arndt, Viktor Frantzen, Oswald Fundneider, Lutz Hagenhaus, Hans Jörg Rothamel, and Lutz Schweitzer. *ISDN The Integrated Services Digital Network: Concepts, Methods, Systems.* Springer-Verlag, Berlin, Germany, 1988.

[BC84] P. Bucciarelli and F. Caneschi. Connectionless services in the OSI reference model. In *Proceedings of the International Computer Communications Conference (ICCC)*, pages 564–569. International Council for Computer Communications, Washington, D.C., 1984.

[BC89] R. Ballart and Y.-C. Ching. SONET: Now it's the standard optical network. *IEEE Communications Magazine*, 27(3):8–15, March 1989.

[BG92] Dimitri P. Bertsekas and Robert G. Gallager. *Data Networks.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, second edition, 1992.

[BGSC92] Pierre E. Boyer, Fabrice M. Guillemin, Michel J. Servel, and Jean-Pierre Coudreuse. Spacing cells protects and enhances utilization of ATM network links. *IEEE Network*, pages 38–49, September 1992.

[BH89] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1:151–160, 1989.

[Bla95] Uyless D. Black. *ATM: Foundation for Broadband Networks.* Prentice Hall Series in Advanced Communication Technologies. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1995.

[Bro92] F.M. Brochin. A cell spacing device for congestion control in ATM networks. *Performance Evaluation*, 16:107–127, 1992.

[BS91] Jaime Jungok Bae and Tatsuya Suda. Survey of traffic control schemes and protocols in ATM networks. *Proceedings of the IEEE*, 79(2):170–189, February 1991.

[CD88] George F. Coulouris and Jean Dollimore. *Distributed Systems: Concepts and Design.* International Computer Science Series. Addison-Wesley Publishing Company, Workingham, England, first edition, 1988.

[Cha83] A. Lyman Chapin. Connections and connectionless data transmission. *Proceedings of the IEEE*, 71(12):1365–1371, December 1983.

[Cha91] C.S. Chang. Smoothing point processes as a means to increase throughput. Technical Report RC 16866, IBM Research Division, May 1991.

[CL91] Xiaoqiang Chen and Ian M. Leslie. A neural network approach towards adaptive congestion control in broadband ATM networks. In

1991 IEEE Globecom, pages 4.2.1–4.2.5. IEEE Communications Society, December 2–5, 1991.

[CLG95] Song Cong, San-qi Li, and Joydeep Ghosh. Predictive dynamic bandwidth allocation for efficient transport of real-time VBR video over ATM. *IEEE Journal on Selected Areas in Communications*, 13(1):12–23, January 1995.

[Com94] The ATM Forum Technical Committee. *ATM User-Network Interface Specification Version 3.1.* The ATM Forum, September, 1994.

[Com95] The ATM Forum Technical Committee. *Traffic Management Specification Version 4.0.* The ATM Forum, August, 1995.

[COO91] J. Cosmos and A. Odinma-Okafor. Characterisation of variable rate video codecs in ATM to a geometrically modulated deterministic process model. In *Queueing, Performance and Control in ATM*, pages 373–380. 13th International Teletraffic Congress, Copenhagen, June 19–26, 1991.

[Dau82] William R. Daumer. Subjective evaluation of several efficient speech coders. *IEEE Transactions on Communications*, COM–30(4):655–662, April 1982.

[Des91] A. Descloux. Stochastic models for ATM switching networks. *IEEE Journal on Selected Areas in Communications*, 9(3):450–457, April 1991.

[DM93] Li Deng and Jon W. Mark. Parameter estimation for Markov modulated Poisson processes via the EM algorithm with time discretization. *Telecommunication Systems*, 1:321–338, 1993.

[DTVV90] M. Decina, T. Toniatti, P. Vaccari, and L. Verri. Bandwidth assignment and virtual cell blocking in ATM networks. In *IEEE Infocom '90*, pages 881–888. IEEE Communications Society, June 3–7, 1990.

[DTW94] Michael Devetsikiotis, J. Keith Townsend, and Mark W. White. Artificial neural netowrks for modeling and simulation of communication systems with nonlinear devices. In *1994 IEEE International Conference on Communications (SUPERCOMM/ICC '94)*, pages 763–768. IEEE Communications Society, May 1–5, 1994.

[ELL90] A. E. Eckberg, Daniel T. Luan, and David M. Lucantioni. Bandwidth management: A congestion control strategy for broadband packet networks — characterizing the throughput-burstiness filter. *Computer Networks and ISDN Systems*, 20:415–423, 1990.

[FDD97] Hany I. Fahmy, George Develokos, and Christos Douligeris. Application of neural networks and machine learning in network design. *IEEE Journal on Selected Areas in Communications*, 15(2):226–237, February 1997.

[FM94] Victor S. Frost and Benjamin Melamed. Traffic modeling for telecommunications networks. *IEEE Communications Magazine*, 32(3):70–81, March 1994.

[FMH93] Wolfgang Fischer and Kathleen S. Meier-Hellstern. The markov-modulated poisson process MMPP cookbook. *Performance Evaluation: An International Journal*, 18(2):149–171, 1993.

[GAN91] Roch Guérin, Hamid Ahmadi, and Mahmoud Naghshineh. Equivalent capacity and its application to bandwidth allocation in high-speed networks. *IEEE Journal on Selected Areas in Communications*, 9(7):968–981, September 1991.

[Gil76] William J. Gilbert. *Modern Algebra with Applications*. John Wiley & Sons, Inc., New York, New York, 1976.

[Gol90] S. Jamaloddin Golestani. Congestion-free transmission of real-time traffic in packet networks. In *Proceedings of IEEE INFOCOM '90*, pages 527–536. IEEE Computer Society, June 1990.

[Gre84] Paul E. Green, Jr. Computer communications: Milestones and prophecies. *IEEE Communications Magazine*, 22(5):49–63, May 1984.

[GRF89] G. Gallassi, G. Rigolio, and L. Fratta. ATM: Bandwidth assignment and bandwidth enforcement policies. In *1989 IEEE Globecom*, pages 49.6.1–49.6.6. IEEE Communications Society, November 27–30, 1989.

[Gro76a] S. Grossberg. Adaptive pattern classification and universal recording I: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23:121–134, 1976.

[Gro76b] S. Grossberg. Adaptive pattern classification and universal recording II: Feedback, expectation, olfaction, and illusions. *Biological Cybernetics*, 23:186–202, 1976.

[Gus90] R. Gusella. A measurement study of diskless workstation traffic on an ethernet. *IEEE Transactions on Communications*, 38(9):1557–1568, September 1990.

[Guy91] I. Guyon. Applications of neural networks to character recognition. In P. S. P. Wang, editor, *Character and Handwriting Recognition: Expanding Frontiers*, World Scientific Series in Computer Science — Volume 30, pages 353–382. World Scientific Publishing Co. Pte. Ltd., River Edge, New Jersey 07661, 1991.

[Hay88] Simon S. Haykin. *Digital Communications.* John Wiley & Sons, Inc., New York, New York, 1988.

[Hay94] Simon S. Haykin. *Neural Networks: A Comprehensive Foundation.* Macmillan College Publishing Company, Inc., New York, New York, 1994.

[HH91] Hilde Hemmer and Per Thomas Huth. Evaluation of policing functions in ATM networks. In *Queueing, Performance and Control in ATM*, pages 111–116. 13^{th} International Teletraffic Congress, Copenhagen, June 19–26, 1991.

[HHS83] R. Hinden, J. Havarty, and A. Sheltzer. The DARPA Internet: Interconnection of heterogeneous computer networks with gateways. *IEEE Computer Magazine*, 21:38–48, September 1983.

[HHS94] Rainer Händel, Manfred N. Huber, and Stefan Schröder. *ATM Networks: Concepts, Protocols, Applications.* Electronic Systems Engineering Series. Addison-Wesley Publishing Company, Workingham, England, second edition, 1994.

[Hir90] Atsushi Hiramatsu. ATM communications network control by neural networks. *IEEE Transactions on Neural Networks*, 1(1):122–130, March 1990.

[Hir91] Atsushi Hiramatsu. Integration of ATM call admission control and link capacity control by distributed neural networks. *IEEE Journal on Selected Areas in Communications*, 9(7):1131–1138, September 1991.

[HL86] Harry Heffes and David M. Lucantoni. A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *IEEE Journal on Selected Areas in Communications*, 4(6):856–868, September 1986.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science*, 79:2554–2558, April 1982.

[HS86] G. E. Hinton and T. J. Sejnowski. Learning and relearning in boltzmann machines. In David E. Rumelhart, James L. M^{c}Clelland, et al., editors, *Parallel Distributed Processing*, Computational Models of Cognition and Perception, chapter 7, pages 282–317. The MIT Press, Cambridge, Massachusetts, 1986.

[Hui88] Joseph Y. Hui. Resource allocation for broadband networks. *IEEE Journal on Selected Areas in Communications*, 9(6):1598–1608, December 1988.

[ITU91a] ITU-T, Geneva. *Recommndation I.113. Vocabulary of Terms for Broadband Aspects of ISDN*, 1991. Revision 1.

[ITU91b] ITU-T, Geneva. *Recommndation I.321. B-ISDN Protocol Reference Model and its Application*, 1991.

[Jai90] Raj Jain. Congestion control in computer networks: Issues and trends. *IEEE Network Magazine*, 4(3):24–30, May 1990.

[Jay86] Nuggehally S. Jayant. Coding speech at low bit rates. *IEEE Spectrum*, 23(8):58–63, August 1986.

[JDSZ95] Sugih Jamin, Peter B. Danzig, Scott Shenker, and Lixia Zhang. A measurement-based admission control algorithm for integrated services packet networks. *ACM SIGCOMM Computer Communication Review*, 25(4):2–13, October 1995.

[JR86] Raj Jain and Shawn A. Routhier. Packet trains — measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, SAC-4(6):982–989, September 1986.

[JV89] P. Joos and W. Verbiest. A statistical bandwidth allocation and usage monitoring algorithm for ATM networks. In *ICC'89*. IEEE Communications Society, 1989.

[Kle75] Leonard Kleinrock. *Queueing Systems*, volume I: Theory. John Wiley & Sons, Inc., New York, New York, 1975.

[KM91] Faouzi Kamoun and M. K. Mehmet Ali. A neural network shortest path algorithm for optimum routing in packet-switched communications networks. In *1991 IEEE Globecom*, pages 4.3.1–4.3.5. IEEE Communications Society, December 2-5, 1991.

[Koh82] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[Kos87] B. Kosko. Adaptive bidirectional associative memories. *Applied Optics*, 26:4947–4960, December 1987.

[Kua94] Lei Kuang. On the variance reduction property of the buffered leaky bucket. *IEEE Transactions on Communications*, 42(9):2670–2671, September 1994.

[Kur93] Jim Kurose. Open issues and challenges in providing quality of service guarantees in high-speed networks. *ACM SIGCOMM Computer Communication Review*, 23(1):6–15, January 1993.

[LBD+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[LD97] Yao-Ching Liu and Christos Douligeris. Rate regulation with feedback controller in ATM networks—a neural network approach. *IEEE Journal on Selected Areas in Communications*, 15(2):200–208, February 1997.

[LeG91] D. J. LeGall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):47–58, April 1991.

[Lid90] William P. Lidinsky. Data communications needs. *IEEE Network Magazine*, 4(2):28–33, March 1990.

[Lip87] Richard P. Lippmann. An introduction to computing with neural nets. *IEEE Acoustics, Speech and Signal Processing (ASSP) Magazine*, 4(1):4–22, April 1987.

[LM94] Robert C. Lehr and Jon W. Mark. On traffic shaping in ATM networks. In *Proceedings of the Singapore International Conference on Communications Systems (ICCS '94)*, volume 2, pages 515–519, Singapre, November 14–November 18 1994.

[LM96] Robert C. Lehr and Jon W. Mark. Traffic classification using neural networks. Presented at *Communication Networks and Neural Networks: The Challenge of Network Intelligence*, March 6–7, 1996.

[LP91] Aurel A. Lazar and Giovanni Pacifici. Control of resources in broadband networks with quality of service quarantees. *IEEE Communications Magazine*, 29(10):66–73, October 1991.

[LTWW94] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[MAS+88] B. Maglaris, D. Anastassiou, P. Sen, G. Karlsson, and J. Robbins. Performance models of statistical multiplexing in packet video communications. *IEEE Transactions on Communications*, 36(7):834–844, July 1988.

[MGF91] José Augusto Suruagy Monteiro, Mario Gerla, and Luigi Fratta. Input rate control for ATM networks. In *Queueing, Performance and Control in ATM*, pages 117–122. 13th International Teletraffic Congress, Copenhagen, June 19–26, 1991.

[MH87] K. S. Meier-Hellstern. A fitting algorithm for markov-modulated poisson processes having two arrival rates. *European Journal of Operational Research*, 29:370–377, 1987.

[Mor91] Robert J. T. Morris. Prospects for neural networks in broadband network resource management. In *Queueing, Performance and Control in ATM*, pages 335–340. 13th International Teletraffic Congress, Copenhagen, June 19–26, 1991.

[MOSM89] Masayuki Murata, Yuji Oie, Tatsuya Suda, and Hideo Miyahara. Analysis of a discrete-time single-server queue with bursty inputs for traffic control in ATM networks. In *1989 IEEE Globecom*, pages 49.5.1–49.5.7. IEEE Communications Society, November 27–30, 1989.

[MP88] Marvin L. Minsky and Seymour Papert. *Perceptrons*. The MIT Press. Cambridge, Massachusetts, 1988.

[MS95] David E. McDysan and Darren L. Spohn. *ATM: Theory and Application*. McGraw-Hill Series on Computer Communications. McGraw-Hill, Inc., New York, New York, 1995.

[MSST91] Tutomu Murase, Hiroshi Suzuki, Shohei Sato, and Takao Takeuchi. A call admission control scheme for ATM networks using a simple quality estimate. *IEEE Journal on Selected Areas in Communications*, 9(9):1461–1470, December 1991.

[MTG93] M. Mittler and P. Tran-Gia. Performance of a neural net scheduler used in packet switching interconnection networks. In *Proceedings of IEEE International Conference on Neural Networks (ICNN '93)*, pages 695–700, 1993.

[NFO89] M. Nomura, T. Fujii, and N. Ohta. Basic characteristics of variable rate video coding in ATM environment. *IEEE Journal on Selected Areas in Communications*, 7(5):752–760, 1989.

[Nie90] Gerd Niestegge. The "leaky bucket" policing method in the ATM (Asynchronous Transfer Mode) network. *International Journal of Digital and Analog Communication Systems*, 3:187–197, 1990.

[Nie93] Gerd Niestegge. The impact of traffic shaping on broadband network performance. *Archiv für Elektronik und Übertragungstechnik (AEÜ)*, 47(5/6):187–197, September/November 1993. *Translation:* International Journal of Electronics and Communications.

[OAT94] Takashi Okuda, Manimaran Anthony, and Yoshiaki Tadokoro. A neural approach to performance evaluation for teletraffic system. In *1994 IEEE International Conference on Communications (SUPERCOMM/ICC '94)*, pages 774–778. IEEE Communications Society, May 1–5, 1994.

[Oht94] Naohisa Ohta. *Packet Viedo: Modeling and Signal Processing.* Artech House Telecommunications Library. Artech House, Inc., 685 Canton Street, Norwood, MA, 02062, 1994.

[OLT92] Teunis Ott, T. V. Lakshman, and Ali Tabatabai. A scheme for smoothing delay-sensitive traffic offered to ATM networks. In *Proceedings of IEEE INFOCOM '92*, volume 2, pages 776–785. IEEE Computer Society, 1992.

[Onv94] Raif O. Onvural. *Asynchronous Transfer Mode Networks: Performance Issues.* Artech House Telecommunications Library. Artech House, Boston, 1994.

[OOM91] T. Okada, H. Ohnishi, and N. Morita. Traffic control in asynchronous transfer mode. *IEEE Communications Magazine*, 29:58–62, September 1991.

[Pao89] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[Pap80] Anthanasios Papoulis. *Circuits and Systems: A Modern Approach.* HRW Series in Electrical and Computer Engineering. Holt, Rinehart and Winston, Inc., New York, New York, 1980. Series Editor: M. E. Van Valkenburg.

[Pap84] Anthanasios Papoulis. *Probability, Random Variables and Stochastic Processes.* Communications and Information Theory. McGraw-Hill Book Company, New York, New York, second edition, 1984. Series Consulting Editor: Stephen W. Director.

[Par94] Craig Partridge. *Gigabit Networking.* Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[PT90] Gurudatta M. Parulkar and Jonathan S. Turner. Towards a framework for high-speed communication in a heterogeneous networking environment. *IEEE Network Magazine*, 4(2):19–27, March 1990.

[Rat91] Erwin P. Rathgeb. Modeling and performance comparison of policing mechanisms for ATM networks. *IEEE Journal on Selected Areas in Communications*, 9(3):325–334, April 1991.

[RF91] G. Rigolio and L. Fratta. Input rate regulation and bandwidth assignment in ATM networks: An integrated approach. In *Queueing, Performance and Control in ATM*, pages 123–128. 13^{th} International Teletraffic Congress, Copenhagen, June 19–26, 1991.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, Computational Models of Cognition and Perception, chapter 8, pages 318–362. The MIT Press, Cambridge, Massachusetts, 1986.

[RVF91a] G. Rigolio, L. Verri, and L. Fratta. Source control and shaping in ATM networks. In *1991 IEEE Globecom*, pages 9.6.1–9.6.5. IEEE Communications Society, December 2–5, 1991.

[RVF91b] G. Rigolio, L. Verri, and L. Fratta. Source control and shaping in ATM networks. In *1991 IEEE Globecom*, pages 9.6.1–9.6.5. IEEE Communications Society, December 2–5, 1991.

[RW90] V. Ramaswami and Walter Willinger. Efficient traffic performance strategies for packet multiplexers. *Computer Networks and ISDN Systems*, 20:401–407, 1990.

[SAG94] G. D. Stamoulis, M. E. Anagnostou, and A. D. Georgantas. Traffic source models for ATM networks: a survey. *Computer Communications*, 17(6):428–438, June 1994.

[Sai92] Hiroshi Saito. Call admission control in an ATM network using upper bound of cell loss probability. *IEEE Transactions on Communications*, 40(9):1512–1521, September 1992.

[Sai94] Hiroshi Saito. *Teletraffic Technologies in ATM Networks*. Artech House Telecommunications Library. Artech House, Boston, 1994.

[Sch87] Mischa Schwartz. *Telecommunication Networks: Protocols, Modeling and Analysis*. Addison-Wesley Series in Electrical and Computer Engineering. Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

[Sch96] Mischa Schwartz. *Broadband Integrated Networks*. Prentice-Hall PTR, Upper Saddle River, New Jersey 07458, 1996.

[SHP91] John D. Spragins, Joseph L. Hammond, and Krzysztof Pawlikowski. *Telecommunications: Protocols and Design*. Addison-Wesley's Telecommunications Library. Addison-Wesley Publishing Company, Reading, Massachusetts, 1991.

[SLCG89] M. Sidi, W. Z. Liu, I. Cidon, and I. Gopal. Congestion control through input rate regulation. In *Proceedings of IEEE Globecom '89.* IEEE Communications Society, November, 1989.

[SMRA89] P. Sen, B. Maglaris, N. Rikli, and D. Anastassiou. Models packet switching of variable-bit-rate video sources. *IEEE Journal on Selected Areas in Communications*, 7(5):865–869, 1989.

[SS91] Hiroshi Saito and Kohei Siomoto. Dynamic call admission control in an ATM networks. *IEEE Journal on Selected Areas in Communications*, 19(7):982–989, September 1991.

[SSD93] Paul Skelly, Mischa Schwartz, and Sudhir Dixit. A histogram-based model for video traffic behaviour in an ATM multiplexor. *IEEE/ACM Transactions on Networking*, 1(4):446–459, August 1993.

[Sta94] William Stallings. *Data and Computer Communications.* Macmillan Publishing Company, New York, New York, fourth edition, 1994.

[SW86] K. Sriram and W. Whitt. Characterizing superposition arrival processes in packet multiplexers for voice and data. *IEEE Journal on Selected Areas in Communications*, 4(6):833–846, September 1986.

[Tan88] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall Software Series. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, second edition, 1988.

[TG92] Ljiljana Trajković and S. Jamaloddin Golestani. Congestion control for multimedia services. *IEEE Network Magazine*, 6(9):20–26, September 1992.

[TGG92] P. Tran-Gia and O. Gropp. Performance of a neural net used as admission controller in ATM systems. In *Proceedings of IEEE Globecom '92*, pages 1303–1309. IEEE Communications Society, December 6–9, 1992.

[TGPM78] Fouad A. Tobagi, Mario Gerla, Richard W. Peebles, and Eric G. Manning. Modeling and measurement techniques in packet communication networks. *Proceedings of the IEEE*, 66(11):1423–1447, November 1978.

[THS94] Ahmed A. Tarraf, Ibrahim W. Habib, and Tarek N. Saadawi. A novel neural network traffic enforcement mechanism for ATM networks. In *1994 IEEE International Conference on Communications (SUPERCOMM/ICC '94)*, pages 779–783. IEEE Communications Society, May 1–5, 1994.

[Tur86] Jonathan S. Turner. New directions in communications (or which way to the information age?). *IEEE Communications Magazine*, 26(10):8–15, October 1986.

[Tur92] Jonathan S. Turner. Managing bandwidth in ATM netowrks with bursty traffic. *IEEE Network Magazine*, 6(5):50–58, September 1992.

[VP89] W. Verbeist and L. Pinnoo. A variable-bit-rate video codec for asynchronous transfer mode networks. *IEEE Journal on Selected Areas in Communications*, 7(5):761–770, 1989.

[Was89] Philip D. Wasserman. *Neural Computing: Theory and Practice.* Van Nostrand Reinhold, New York, New York, 1989.

[WH91] E. Wallmeier and C. M. Hauber. Blocking probabilities in ATM pipes controlled by a connection acceptance algorithm based on mean and peak bit rates. In *Queueing, Performance and Control in ATM*, pages 137–142. 13^{th} International Teletraffic Congress, Copenhagen, June 19–26, 1991.

[WL90] Bernard Widrow and Michael A. Lehr. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, September 1990.

[WM93] Guoliang Wu and Jon W. Mark. Discrete time analysis of leaky-bucket congestion control. *Computer Networks and ISDN Systems*, 26:79–94, 1993. Special issue on Teletraffic Issues in ATM Networks.

[Wol89] Ronald W. Wolff. *Stochastic Modeling and the Theory of Queues.* Prentice Hall International Series in Industrial and Systems Engineering. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1989. Series Editors: W. J. Fabrycky and J. H. Mize.

[WRR88] G.M. Woodruff, R.G.H. Rogers, and P.S. Richards. A congestion control framework for high-speed integrated packetized transport. In *1988 IEEE Globecomm*, pages 7.1.1–7.1.5. IEEE Communications Society, November 28–December 1 1988.

[WT90] David J. Wright and Michael To. Telecommunication applications of the 1990s and their transport requirements. *IEEE Network Magazine*, 4(2):24–40, March 1990.

[YHS96] S. A. Youssef, I. W. Habib, and T. N. Saadawi. A neural network control for effective admission control in ATM networks. In *Proceedings of IEEE International Conference on Communications (ICC '96)*, pages 434–438. IEEE Communications Society, June 23–27, 1996.

[YS91] Hiroshi Yamada and Shuichi Sumita. A traffic measurement method and its application for cell loss probability estimation in ATM networks. *IEEE Journal on Selected Areas in Communications*, 9(3):315–324, April 1991.