# Using Fuzzy Logic for Congestion Control at the User-Network Interface of Asynchronous Transfer Mode Networks

by

Frederick Mark Albrecht

Submitted to the University of Cape Town in
fulfilment of the requirements for the degree of
Masters of Science in Electrical Engineering.

June 1997

# SYNOPSIS

The aim of this thesis was to design a fuzzy logic controller that could be used to prevent congestion in Asynchronous Transfer Mode networks. The controller would be located at the User-Network Interface of the network and would be used to monitor the incoming traffic. The control action would be to allow conforming traffic through to the network unmodified, while tagging or dropping non-conforming traffic.

Three fuzzy logic controllers are presented, the latter two being improvements on design(s) that preceded it. All three controllers are designed to police the sustainable cell rate of Variable Bit Rate (VBR) traffic sources, and take the Sustainable Cell Rate and Peak Cell Rate as input Parameters. The first two controllers require an additional parameter called the Burst Threshold, while the third controller takes the Initial Burst Size as its third traffic parameter. The controllers then continuously analyse the state of the traffic by applying a set of rules that it was given and decides whether the source is conforming or non-conforming, taking corrective action if required.

The first controller included two buffers that were used to determine the state of the connection. It was however found that it would be too difficult to determine what sizes to make those buffers, and therefore that design was abandoned in favour of the second controller.

The second controller eliminated the need for buffers in the design, while still being able to tag or drop non-conforming cells and allowing conforming cells through unmodified. This was done by including a token leaky bucket in the design and modifying the fuzzy sets and rule base of the fuzzy logic engine accordingly.

A sinusoidally varying VBR source was chosen to test the performance of the second controller. The entire design was then simulated using the OPNET® network simulator. The simulation results showed that the controller allowed the traffic through if the source was conforming to its traffic contract. In the case where the source was conforming most of the time, the controller could distinguish between dropping and tagging non-conforming cells, while allowing cells through when the source returned to its conforming state. It was found that the controller dropped all of the cells if the

source was operating well outside of its traffic parameters. In the last case the results also showed that the GCRA did not drop all of its incoming traffic, but allowed the negotiated amount of cells through while dropping the rest. Although it is up to the network to decide how to handle non-conforming traffic, it was felt that the source would receive a better quality of service if the GCRA was used as the policing algorithm instead of the fuzzy logic controller. The second fuzzy logic controller was thus not much better than the GCRA although it was performing to its specifications. A third controller was then designed to see if it could be possible to improve on the GCRA.

The third controller modified its predecessor from a rate based controller to a window based controller. Here only a certain number of cells are allowed through during the window period, where this number of cells is also referred to as the credit that the source owns. In addition to this, the amount of credit that the source owns is increased if the source is conforming, and is decreased if it is non-conforming. This enabled the controller to accept a large burst of cells if the source has been conforming for some time.

Simulations were performed where a model of a MPEG-encoded video stream was used as a traffic source. The results showed that the GCRA dropped more cells than the final fuzzy logic controller even in the case when the source was conforming. The fuzzy logic controller only dropped cells during the start of the connection if the specified Initial Burst Size was too small. The GCRA on the other hand dropped cells throughout the duration of the connection. This showed that the GCRA was not suited to police the sustainable cell rate in real time *without* being over-dimensioned. When the source was non-conforming it was found that the performance of the fuzzy logic controller tended towards that of the GCRA.

It was found that the final fuzzy logic controller performed better than the GCRA when policing an MPEG-encoded video source.

# ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to the following people, whom have assisted me, directly and indirectly, in the completion of my thesis:

My family, for their tremendous understanding and moral support. Especially for providing me with the space to finish this dissertation.

My Mom, for believing in me and not being afraid to say so.

My Dad, for helping and proof-reading my work.

My supervisor, M. J. E. Ventura, lecturer in the Department of Electrical and Electronic Engineering at the University of Cape Town (UCT), who provided continuing academic insight, and was always available to provide constructive assistance.

Telkom South Africa for their support, and providing me with the opportunity the get hands-on experience with the ATM equipment in the Communications Research Lab at UCT.

# *Dedication*

*The future is yours, with all its mystery and promise.*
*Walk through life's winding pathways with courage and humour and hope,*
*savouring fully your triumphs and learning from your trials.*
*Strive to live to the best of your ability,*
*by those personal standards of honour and excellence,*
*which only you can set.*
*Cherish above all your individuality.*
*Realise always that you are capable of achieving the only kind of success that really matters,*
*that of being at peace with yourself and the world around you.*
*Your moment has come, the world awaits, the future is yours.*

*- Author unknown -*

*I dedicate this to my godson, Jayde Darren Pullen, a five year old boy who showed me something that changed my life. When you are old enough to understand, Jayde, I hope to return the favour.*

*And thank you, Big Brother, for showing me the Mystery in Life.*

# TABLE OF CONTENTS

# LIST OF FIGURES

**Page**

# LIST OF TABLES

**Page**

# 1. INTRODUCTION

Asynchronous Transfer Mode (ATM) networks are promising to be the networks of the future, capable of handling all types of services foreseen for Broadband Integrated Services Networks (B-ISDN). But in order to ensure its users of a reliable communication and data service, ATM networks will have to provide those users with a good quality of service. Having all of these services on the same network is a big advantage from a users' point of view, but from a traffic management point of view the problem arises on how to control all of these different, and sometimes conflicting, services.

Traffic managers are faced with the question of how to prevent the misuse or abuse of resources in the network. Traffic violations have a negative impact on the operation of the network, and hence it is accepted that some form of Traffic Control is needed.

Before we consider what the form of such control should be, lets first determine what desirable features an ideal controller should include. The ITU-T recommends the following features in their I.371 Recommendation:

1. rapid response time when the traffic contract is violated,
2. detection of any illegal traffic, and
3. simplicity of design.

Several control methods have been proposed in the literature. Among these are the Leaky Bucket and Token Bucket algorithms, and also the more traditional windowing flow control methods, of which the Jumping Window algorithm is one. Although popular they have been shown to be inefficient at the control of the conflicting requirements of ideal policing [ 2 ].

The mathematical and classical control theory approaches to controller design can also be considered, but they too have their drawbacks. In the mathematical approach the core of the design process is coming up with a mathematical model of the traffic in the connection and then to analyse that traffic model [ 1 ][ 2 ]. This is generally a difficult

process due to the complexity of the mathematics involved when deriving the equations, and often these equations have to be simplified when trying to do policing using the derived mathematical models [ 3 ]. The classical control theory approach is also mathematically intensive. The designer of a controller using this approach needs to have a thorough understanding of both the design methods applied in classical control as well as the dynamic properties of the system that needs to be controlled [ 4 ]. This process can be as complex as the mathematical approach mentioned above, if not more so, and as such is even more inappropriate.

These control methods show that it is difficult to come up with a controller that is able to predict and control the rate at which cells[1] are generated [ 5 ][ 6 ][ 7 ]. This is mostly due to the many different types of traffic sources and the varying parameter values (especially for real-time bursty sources such as video) that can be associated with each one. New methods for designing traffic controllers are thus needed if one ever hopes to approach the performance of the ideal controller.

Researchers are now exploring the use of Artificial Intelligence (AI) in the design of traffic controllers. The assumption from which they proceed is that if some form of artificial intelligence can be built into the controllers, then those controllers will be better equipped to perform the tasks they were assigned. Most importantly, the advances in technology, especially the raw computational power of today's central processing units (CPU's), now makes it possible to implement these processes in real-time, and thus making them practical.

Neural Networks and Fuzzy Logic are two of the methods that are commonly used to implement AI. The first mimics the operation of neurons in the human brain by being capable of "learning," while the second mimics the decision making process of the human brain when evaluating a problem by applying pre-learnt knowledge and linguistic descriptions of the problem at hand. Both methods allow the user to step back from having to understand the complexity of the mechanics of the control system's design, and to instead concentrate on what decisions the controller has to make in order to operate effectively. The use of such design methods presents one with a more intuitive approach to designing traffic controllers.

[1] ATM network packets.

operate effectively. The use of such design methods presents one with a more intuitive approach to designing traffic controllers.

This thesis approaches the control problem from an Artificial Intelligence point of view, more specifically, how Fuzzy Logic can be used to design a preventive congestion controller for traffic entering the User-Network Interface in ATM networks. Fuzzy Logic was chosen because it allows the control problem to be expressed linguistically, rather than mathematically, leaving the mathematics to be handled by the fuzzy set theory [ 8 ], and thereby simplifying the design process. By using Fuzzy Logic, the state of the connection can be described in simple human terms, decisions can be made the way humans would make them, and expert knowledge about the controlled process can be built directly into the controller itself.

## 1.1 Background

Traffic Management can be divided into two categories: Connection Admission Control (CAC) and Usage Parameter Control (UPC). CAC takes place at call set-up when the network has to consider whether to allow or reject a new connection, while UPC monitors the traffic along the connection once the connection is established.

A traffic controller needs to know what type of traffic a connection will be carrying before it can decide whether or not to accept that connection. The process that requested the connection therefore has to inform the controller at call set-up what type of service it requires by supplying the traffic parameters that describe that connection. ITU-T recommendation I.371 recommends that the Sustainable Cell Rate (SCR), Peak Cell Rate (PCR), Burst Tolerance[2] (BT), and the Cell Delay Variation Tolerance (CDVT) are required parameters when specifying the traffic contract. The PCR and CDVT must be specified for every connection, while Variable Bit Rate (VBR) services may also need to specify SCR and BT. This process of negotiating a connection with the network is called Connection Acceptance Control and is necessary so that the network can determine whether it can accept the new connection without degrading the quality of service of existing connections.

Once the network accepts the connection, it then constantly has to monitor that the current traffic parameters stay within the parameters agreed upon at call set-up time; taking some form of corrective action when those parameters are exceeded. This process of constant monitoring is called Usage Parameter Control, and will be the focus of this thesis.

Several control methods (and some of their short comings) were outlined above, and the suggestion made that alternate control methods be sought to implement better ones. Several designs were found in the literature that used Fuzzy Logic in their implementation, and all of them report improvements on the traditional control methods [ 9 ][ 10 ][ 11 ]. They also show that the use of Fuzzy Logic as a control method is valid and brings us closer to the aim of realising the ideal policing function.

[2] Also known as the Burst Size, and is related to the Burst Duration.

## 1.2 The Thesis Objective

The aim of this thesis is to use Fuzzy Logic to implement a Congestion Controller at the User-Network Interface of ATM networks. The Fuzzy Logic Controller (FLC) that is the result of this alternate design process, should be able to apply the rules that it was given and decide what the state of the connection is. After evaluating the state of the system it then has to generate a control action that should:

1. allow conforming cells through to the network,
2. drop cells that are totally non-conforming, and
3. optionally tag cells that are close to being defined as non-conforming before allowing them through to the network.

The design should be simple and respond quickly when the traffic contract is violated. Simply put, it must try to emulate the ideal controller.

Once completed, the design then has to be simulated to see how it performs. OPNET®, a network simulator, was chosen for its ability to handle packet switched networks, and ATM networks in particular. OPNET® on its own, however, does not have the modules that allow the user to handle fuzzy logic. Instead it allows the user to write custom C code that can be included in the simulation environment. This allows the user to add any functionality to the simulator that the standalone package lacks. This meant that a fuzzy logic engine could be included if source code for such an engine could be obtained.

Several fuzzy logic engines were found on the Internet, but only Fuzzy CLIPS had the capabilities that were needed for this project as well as being distributed with its source code. After evaluating Fuzzy CLIPS the author found that there was simply too much source code to become familiar with before it could be incorporated into OPNET®. So, after some consideration the author decided to write a new fuzzy logic engine and included that into OPNET®. This new engine is less complex than Fuzzy CLIPS, and has exactly the functionality that the author required for this design. The new engine is called "Fuz."

## 1.3 Dissertation structure

The outline of the rest of the thesis is now presented. Chapter 2 introduces the reader to this new technology called ATM and shows how policing is currently performed. The design of the Fuzzy Logic Congestion Controller (FLCC) is presented in stages in Chapter 3 so that the reader can get an overall view of what prompted the need for the modules implemented in the design. The simulation of the design is presented in Chapter 4 as well as the results obtained by using the OPNET® Network Simulator. Chapter 5 presents an improved version of the controller, and is used to police an MPEG-encoded video source. The simulation results are presented in Chapter 6 where they are compared with the Generic Cell Rate Algorithm. Lastly, Chapter 7 presents conclusions and recommendations for further study.

The author realises that a discussion of the use of Fuzzy Logic to design anything, can not be complete without providing the reader with a basic description of how fuzzy logic works. But since this thesis is not on the workings of fuzzy logic, but rather on how one can design controllers with it, Appendix A presents a fuzzy logic primer that leads the reader through the design process of a rather simple controller. The design is not a traffic controller, and is presented to give the reader a more "hands-on" way to consolidate the theoretical and practical aspects of fuzzy logic. It was decided to present this information as an appendix so that the flow of information through the various chapters is not broken for readers who do not need this information.

Appendix B describes Fuz, the Fuzzy Logic Engine that was developed during the course of the thesis, and how it integrates with OPNET®. The reader is shown how fuzzy variables and their associated fuzzy sets are defined in the system, and how the rule base is stored. The source code for Fuz is supplied on the disk that accompanies this write-up.

Appendix C presents graphs that were considered to be too detailed for inclusion in Chapter 6.

Appendix D presents how to configure OPNET in order to run the simulations.

Appendix E presents the OPNET processes and the detailed descriptions of those processes that make up the simulation. It presents the states of each process and the source code that is used in each one.

# 2. POLICING IN ATM

ATM is a new technology that promises to be the transport medium for the computer networks of the future, and is the next step in the evolution of today's Integrated-Services Digital Networks (ISDN). It is designed to work at extremely high transmission rates, and as such is also referred to as the transport medium for Broadband ISDN (B-ISDN).

ATM networks, their switches, and the ATM cell packet format, have been designed to provide this high transmission rate. The ATM User-Network Interface Specification version 3.0 provides specifications for the physical layer ATM interfaces for both the public and private User-Network Interfaces, and at the moment 44.736 Mbps, 100 Mbps and two 155.52 Mbps interfaces have been specified.

At theses high transmission rates, however, one finds that there is a lot of data present simultaneously on a transmission line when considering the long distances over which ATM connections are normally used. For example, data that is being sent at a rate of 155 Mbps over a 300 km optic fibre link can have a maximum of 155 Kb of data in transit between the source and the termination point (or destination) at any particular point in time. This means that if any problem happens to occur at the source, then the destination only becomes aware of the situation after 155 Kb of data has been received. In the case where the source has been transmitting at too high a rate, 155 Kb of "illegal" traffic will be present on the connection that could have been used by another traffic source.

This brings one to why policing is required in ATM networks. Two of the most important points that policing in ATM attempt to provide is the insurance that:

1. incoming traffic will not flood the network, and
2. all users will be treated fairly by the network.

The first point aims to prevent the buffers in the network from overflowing. This is a necessary item because when buffers start overflowing it means that no more cells are

stored and results in cells being dropped. This affects the performance of all traffic arriving at the particular buffer and degrades the performance of the network.

The second point aims to ensure the user that if it keeps its traffic within the limits that were agreed upon by both the user and the network, then the network will do its utmost to get the user's traffic through to its destination. This also implies that if a user (i.e. traffic source) exceeds its traffic limits, then it must not be allowed to affect the quality of service of other users who share the same network resources.

ATM also supports many different classes of services, including voice, video and data services, real-time and non-real-time services. One finds that these different classes of services all have different traffic parameters, and even in the same class of service one can find traffic parameters differ as well.

This large variety of traffic sources and their traffic parameters make it difficult to come up with a universal congestion control strategy, and intense research is currently being done by many parties to try and solve this problem. Accordingly the ITU-T has specified an initial set of traffic congestion control guidelines in their I.371 specification. The ATM-Forum has also presented a more advanced version of this specification that is known as the ATM User-Network Interface Specification version 3.0. These specifications are however only guidelines. Network providers are not obliged to implement them, and one finds that no accepted fixed strategy has yet been adopted.

It is difficult to come up with a congestion controller that is able to predict and control the rate at which cells are generated. ATM sources can range from a few Kbps to several hundred Mbps and one finds that most simple congestion control schemes end up by penalising either one or the other end of that spectrum.

Several types of traffic control methods exist in order to prevent congestion. The ATM-Forum's ATM UNI Specification version 3.0 presents the following categories:

1. Network resource management.
2. Connection admission control.
3. Usage Parameter control.
4. Priority control.

5. Traffic shaping.

These categories are suggested to *prevent* congestion in the network. The UNI specification suggests the following to *reduce* or *eliminate* congestion once it has set in:

1. Selective cell discarding.
2. Explicit forward congestion indication.

This chapter discusses these different types of traffic and congestion control methods that are used in ATM networks, and touches on the search for alternate control methods.

## 2.1 Traffic Control Methods

### 2.1.1 Network Resource Management

The idea behind Network Resource Management (NRM) is that virtual paths can be used to simplify the management of resources in the network. This is done by grouping Virtual Channel Connections (VCC) of a similar type together into a Virtual Path Connection (VPC) so that one control strategy can be used on all of them.

Several things are simplified with this control method:

1. *Call Admission Control (CAC) and Usage Parameter Control (UPC)*: The same CAC and UPC algorithm can be used on all the VCCs in the VPC, and this reduces the complexity of the system.
2. *Message Distribution*: A single message can be distributed to all the VCCs in the VPC when congestion occurs.
3. *Bandwidth Reservation*: A VPC has a certain amount of bandwidth reserved for it, and therefore the network does not have to consider the entire UNI when a new VCC is required inside a VPC. Only the VPC of interest needs to be consulted. This method is also simplified if the requirements of each VCC are set to that of the most demanding VCC in the VPC.

Lastly, since a VPC bundles VCCs into a group, they experience similar conditions along their path through the network (delay, congestion, etc.).

### 2.1.2 Connection Admission Control

A Traffic Controller needs to know what type of traffic the network is carrying in order to properly control that traffic, and therefor it needs to know the parameters that describe the traffic. Thus at call set-up time the traffic source has to apply for the type of service required, supplying traffic parameters such as the average bit rate, peak bit rate, burstiness, etc. before access is allowed into the network. This process of negotiation is called Connection Acceptance Control (CAC).

CAC is one of the first control methods that are used to prevent congestion control. It decides whether or not enough resources are available inside the network to support the new connection. A new connection is only accepted if enough resources are available and if the new connection will not impinge on the quality of service that current connections are receiving. The network then agrees to provide the negotiated quality of service as long as the connection stays within its traffic parameters.

### 2.1.3 Usage Parameter Control

Once the network accepts the connection it then constantly has to monitor that the source keeps his traffic within the parameters agreed upon at call set-up time, taking some form of corrective action when those parameters are exceeded. This process is called Usage Parameter Control (UPC).

As mentioned in the introduction, the ITU-T recommends the following features for a UPC function in their I.371 Recommendation:

1. Rapid response time when the traffic contract is violated,
2. Detection of any illegal traffic.
3. Simplicity of design.

It also presents that the UPC control action should be to:

1. Pass conforming cells.
2. Optionally tag cells[3].
3. Discard non-conforming cells.

UPC polices all connections entering the UNI, including checking that the Virtual Path Identifier (VPI) and Virtual Channel Identifier (VCI) of incoming traffic is valid. VPIs and VCIs are checked to make sure that traffic from an invalid source are not accepted into the network. This not only protects the network from overload, but also provides a form of security from unwanted sources.

[3] Only works on previously untagged (CLP=0) cells.

The location of the UPC function depends on the configuration of the connection. Three cases were taken from the UNI Version 3.0 specification and are presented in Figure 2-1.

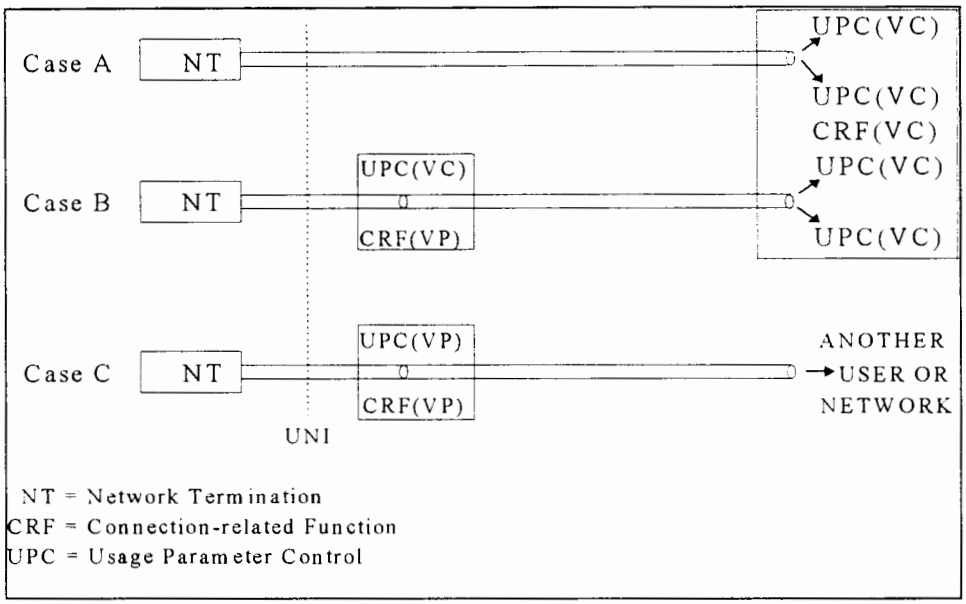



Figure 2-1 Location of the Usage Parameter Control.

Case A shows that if a connection is connected directly to a Virtual Channel Connection Related Function (CRF(VC)) then the UPC function must be performed before the switching function is executed.

Case B shows that if a connection is directly connected to a Virtual Channel Connection Related Function (CRF(VC)) via a Virtual Path Connection Related Function (CRF(VP)) then the UPC function must be performed on the VCCs in CRF(VC)s and VPCs CRF(VP)s.

Case C shows that if the connection is to another user or network provider via a CRF(VP) then the UPC function must be performed on VPCs only. The first network provider that provides CRF(VC) must then implement the UPC function for VCCs.

*2.1.3.1 The Generic Cell Rate Algorithm*

The I.371 and UNI 3.0 presents an algorithm called the Generic Cell Rate Algorithm (GCRA), that can be used to monitor the peak cell rate and the sustainable cell rate of a connection. It is also referred to as the Virtual Scheduling Algorithm and the Continuous-State Leaky Bucket Algorithm.

```
/*
    The GCRA
    Call with parameters I (increment) and L (limit)
    Returns 0 if non-conforming, 1 if conforming.
*/
TYPE TAT=0;

int GCRA(TYPE I, TYPE L)
{
    int is_conforming = 0;

    TYPE ta = current_time;

    if (TAT < ta)
    {
        TAT = ta;
        TAT = TAT + I;
        is_conforming = 1;
    }
    else
    {
        if (TAT < (ta + L))
        {
          TAT = TAT + I;
          is_conforming = 1;
        }
    }

    return is_conforming;
}
```

Figure 2-2 The Generic Cell Rate Algorithm.

Figure 2-2 presents pseudo-C code to implement the GCRA and must be executed every time a new cell arrives. The function takes two parameters, an Increment value I and a Limit value L, and then calculates whether the cell (that has just arrived) arrived before or after its theoretical arrival time (TAT). If the cell arrived on or after the TAT, then the function returns a 1 to indicate that the cell is conforming. Otherwise it returns

a 0 to indicate that the cell is non-conforming. The GCRA function is initialised at the arrival of the first cell by setting the TAT to the time when the cell arrived.

The GCRA is called with $I=T_p$ and L=CDVT when used to police the PCR, where $T_p$ is set to the inverse of the negotiated PCR.

When policing the SCR, the GCRA is called with $I=T_s$ and $L=\tau_s$. $T_s$ is set to the inverse of the negotiated SCR, while $\tau_s$ is set to

$$\tau_s = (MBS - 1) * (T_s - T_p) \quad (1)$$

where MBS is the negotiated Maximum Burst Size[4].

### 2.1.4 Priority Control

The objective of Priority control is to discard lower priority cells before dropping higher priority cells. This is done to protect the performance of the higher priority traffic.

### 2.1.5 Traffic Shaping

Traffic shaping is used to space cells by altering the traffic characteristics of the source to achieve a desired traffic characteristic. The traffic shaper may drop cells that are non-conforming, but should maintain the sequence that the cells arrived in, i.e. it should work on a First-In-First-Out (FIFO) basis.

The plain leaky bucket is an example of this method. Cells that arrive at the UPC function are stored in a FIFO buffer and are let out at a constant rate. The cell is dropped if it arrives at a full buffer. Cells exit the bucket at the negotiated rate.

Another example is the token leaky bucket. This method uses another buffer to store tokens that are generated at the negotiated rate. A cell is allowed to exit the cell buffer if there is a token available in the token buffer. A token is taken out of the token buffer for each cell that is allowed to exit.

---

[4] See [ 22 ] for a derivation of $\tau_s$.

The plain leaky bucket produces cells at the negotiated rate, while the token leaky bucket has the additional functionality of allowing a burst of cells through as long as there are tokens left. When the tokens run out it conceptually reverts back to the plain leaky bucket by only allowing cells through at the negotiated rate.

### 2.1.6 Fast Resource Management

This is the last of the traffic control methods. This method enables the source to request the network to temporarily allow it to exceed its traffic parameters (or traffic contract). The network may concede to this request if it determines that there is sufficient capacity available to allow the increased traffic load, and returns the connection to its previous state once the temporary period has expired

## 2.2 Congestion Control

The UNI 3.0 specifies that Selective Cell Discarding and Explicit Forward Congestion indication have been defined to enable the network to take corrective action when in a state of congestion.

### 2.2.1 Selective Cell Discarding

The Cell Loss Priority (CLP) bit in the ATM cell header sets the priority of a cell. If CLP=0 then the cell has a high priority, otherwise it has a low priority. Cells that are tagged have their CLP=1, and may be discarded by the network to avoid congestion. However, to recover from congestion the network may drop both CLP=1 and CLP=0 cells. This latter action (dropping CLP=0 cells) is justified only in the case when the source is non-conforming.

### 2.2.2 Explicit Forward Congestion Control

Any node in the network may set the forward congestion indication in the cell header if it experiences congestion. The forward congestion indication may however not be cleared once set. In this way the destination can be informed if congestion occurred along the path of the cell. This information may then be used to initiate congestion control functions in the same direction as the received cell, e.g. space the cells out using a leaky bucket, in order to lessen the effects of congestion.

## 2.3 The Search for Alternate UPC Methods

The problem with coming up with new UPC methods lies in that it is difficult to come up with a model of an ATM network that takes into account all the various important characteristics of the traffic in the network [ 5 ]. And when one does come up with such a model, it is generally mathematically too complex to analyse to any significant degree. Classical Control Theory, for example, is unsuitable when trying to design a congestion controller in an ATM environment, because one needs to understand the dynamics of the system before one can design a controller for it. Alternate control methods are thus needed to implement UPC.

The most popular UPC methods are:

1. Flow Control.
2. Rate Control.
3. Window Control.
4. Artificial Intelligence.

Flow control methods handle congestion by providing feedback to the source on whether its traffic is conforming or not, and whether the network is experiencing difficulty in handling the traffic. When congestion sets in the source is then instructed to lower its traffic rate, and is often not allowed to increase its rate until it is granted permission to do so, implicitly or explicitly.

This type of control method is not suitable for all types of traffic sources in ATM networks because of the high data rates and the long distances over which the traffic has to be transmitted. Feedback under these conditions is too slow when trying to reduce the cell transmission rate, when compared with the propagation delays across the network. In other words, simply too much data is transmitted before the source receives the message to reduce its traffic rate. Flow control has not had much success in controlling Constant and Variable Bit Rate services for this reason, but is an integral part of controlling Available Bit Rate services [ 23 ][ 24 ][ 25].

Rate control differs from flow control in that rate control is generally applied at the interface of the network, while flow control is spread across the network. Rate control aims to make sure that no non-conforming traffic is allowed into the network in the first place, thereby keeping unwanted traffic from using up valuable resources in the network. Cells are tested on a cell by cell basis, and are checked in real-time to make sure that they conform to their traffic contract. It is thus important in this context that the traffic parameters which are selected to describe the source *can* be monitored in real time. Issues dealing with monitoring an average value (such as the mean cell rate for duration of the connection) should be handled carefully because the "overall" average value can not be determined on a per cell basis. Often an additional variable, such as the allowable burst size, or a substitute variable, such as the current average rate, is used to provide "instantaneous" values that can be used on a per cell basis.

The GCRA is one method that is commonly used to implement rate control: it is simple to implement and can be used on a per cell basis. But because of the difficulty of monitoring average values using a rate based method, other methods are also available that may be used in place of it.

Window based control methods are used to move away from having to evaluate the source on a per cell basis. Here a period is set aside during which a certain amount of cells are allowed through the network. Once the allotted number of cells have been allowed through, the rest are either dropped or tagged depending on the discretion of the network. The start of the next window period then resets the number of cells that may be allowed through.

Depending on the type of window method being used, window based control methods can also change the number of cells that it allows through. For example, if the source had been non-conforming for some time then its allotted number of cells may be reduced, and if it has been compliant then more cells may be allotted to it. Various window based methods exist, ranging from the simple Jumping Window algorithm to the more complex Exponentially Waited Moving Average algorithm. All have their good and bad points. Please refer to Rathgeb in [ 2 ] for a comparison of their performance.

The new trend in trying to develop new control methods is to use Artificial Intelligence (AI) to decide whether a source is conforming or not. The idea is that if one can build a measure of intelligence into the control method then it will perform better. AI presents the UPC designer with the ability to teach or train the controllers in the switches, and also to build existing knowledge into them. This helps to bring the designer and the expert of the system closer together in the design process, opening whole new possibilities in UPC design.

Fuzzy Logic and Neural networks are two of the most popular methods used to implement AI in ATM networks, and several designs have been implemented in the literature that show that such designs, although in their infancy, have merit. Please refer to [ 10 ], [ 11 ], [ 12 ], and [ 15 ] for designs using Fuzzy Logic, and [ 26 ], [ 27 ], [ 28 ], [ 29 ], and [ 30 ] for those using Neural Networks.

One of the problems that was mentioned earlier was that it is difficult to come up with a mathematical model of traffic in an ATM network. This is where Neural Networks outperform the mathematical approach. Designs have been shown [ 27 ] that can learn to predict the traffic parameters of traffic sources, and even the predict the behaviour of the source itself [ 27 ], [ 31 ]. Scheffer et al presented a similar approach in [ 32 ] for modelling traffic using Fuzzy Logic. This ultimately means that controllers no longer have to depend on mathematical derivations to characterise the process to be controlled, but can *learn* the process' behaviour instead. And once the control model is available, the control method is generally not far off.

## 2.4 In Conclusion

One finds that policing in ATM has many challenges, and that many parties are actively busy finding solutions to these challenges. This chapter presented the framework from which these people are working, outlined the solutions they have come up with so far, and gave an indication of where they are heading.

Humans are ingenious creatures, and it is this author's view that we will make this new technology work.

# 3. THE FUZZY LOGIC CONTROLLER

Many different service categories have been specified for ATM traffic. Among these Constant Bit Rate (CBR), Variable Bit Rate (VBR) and Available Bit Rate (ABR) services are considered to be the most important. Here one is presented with three services that differ greatly in their traffic characteristics and the type of services they are suited for.

CBR services are used when a traffic source transmits at a known peak rate, and keeps on transmitting at that rate. This type of traffic sources can be found in today's telephone systems where most of the voice-grade PCM channels, and T1 circuits, etc., use constant-rate, synchronous bit transmission. The inclusion of CBR allows a smooth transition from today's telephone systems to the B-ISDN system of the future.

VBR is used when a traffic source has a peak transmission rate, but on average has a transmission rate lower than that peak rate. This type of traffic is generated by sources such as real-time video conferencing and multimedia services. MPEG compression schemes are commonly used with these services, and because of the way that MPEG works, the transmission rate after compression can vary greatly. VBR is more suited to this type of traffic than CBR since most of the time the transmission rate will be below the peak rate, thus allowing for better channel utilisation.

ABR services allow even greater flexibility when specifying the type of traffic. With ABR it is possible to specify that the network must provide a minimum transmission rate of 10 Mbps for a connection, and that the connection might have a peak of 20 Mbps. The network is obliged by its contract to guarantee the 10 Mbps rate, and does its best to provide double the rate, i.e. 20 Mbps, when it is needed. There is however no guarantee that the 20 Mbps rate will be provided. This type of service is suited (but not limited) to, for example, a company that connects its offices via leased lines. The company might choose to specify enough capacity for the minimum foreseen traffic load, and then specify an upper bound on the *possible* peak load. Specifying the traffic contract in this manner means that the company is no longer forced to keep a connection (as in VBR) that will very seldom be used to capacity. This means that it

would cost them less if they agree that their traffic might be lost (or that congestion can occur) during their busiest period of the day.

ABR is the only one of these three services where the network provides the source with feedback on how its traffic is affecting the traffic load inside the network. With this information the source can then reduce its transmission rate until such time that enough bandwidth is again available to handle its traffic. This type of flow control takes place inside the network, where any of the network nodes might determine that a reduction in the source rate is needed. CBR and VBR policing generally only take place at the interfaces to the network, i.e. at the UNI, and does not provide the source with feedback on the effects its traffic has on the network. Any non-conforming traffic is dealt with appropriately, *before* it is allowed into the network, putting less strain on the resources inside the network in the process.

This thesis concentrates on controlling the traffic before it enters the network, and as such is only suited to police CBR and VBR sources. The controlling mechanism, that will be described below, is a rate controller that attempts to prevent congestion inside the network by controlling the rate at which traffic is allowed to enter the network. ABR traffic will not be considered at all because its flow control scheme is spread along the path of the connection, and not just at its starting point. Thus the controller does not provide feedback to its traffic source as would have been needed to handle ABR traffic a well.

The following sections describe the traffic parameters that were considered necessary to control CBR and VBR sources, and how they interrelate with the modules that are needed in the proposed preventive congestion controller. Work done by other researchers who also used fuzzy logic in their designs, is also discussed to show how their designs influenced the final design of the controller that is presented in this chapter. This then leads on to the actual presentation of the design of the Fuzzy Logic Congestion Controller that is used to control these traffic sources.

## 3.1 The Traffic Parameters

The first stage of the design is to decide what traffic parameters sufficiently describe the characteristics of the source. These traffic parameters can then be used as inputs to the controller that will be used to prevent congestion. The ITU-T Recommendation I.371 recommends that the four following parameters be used:

1. the Sustainable Cell Rate (SCR),
2. the Peak Cell Rate (PCR),
3. the Burst Tolerance (BT), and
4. the Cell Delay Variation Tolerance (CDVT).

When considering CBR and VBR services, the PCR and CDVT must be specified for every connection, while Variable Bit Rate (VBR) services may also need to specify SCR and BT. One should note that the PCR is a mandatory source traffic parameter, while SCR and BT are optional. CDVT is however not a source traffic parameter, but is a result of the cell clumping phenomenon that is due to the slotted nature of ATM, the physical layer overhead and the ATM layer functions. Since CDVT is not caused by the source but may always be present due to the architecture of ATM, the network has to inform the user of the set of values for CDVT that the network supports. The source must then choose a value for its CDVT that can be tolerated by the traffic that it will generate. Figure 3-1 shows the reference configuration and equivalent terminal for the definition of the peak cell rate of an ATM connection (taken from I.371). It shows the CDVT originates between the Physical Layer Service Access Point and the Public UNI.

Although the SCR and BT are optional parameters according to I.371, the author feels that they are necessary when trying to control VBR traffic. The SCR gives an indication of the average rate of the connection, and in a way reflects the history of the connection since it is defined over the duration of the connection, while the BT determines how long a VBR service is allowed to transmit at its peak rate. Simply supplying the PCR on its own would mean that the network would have to provide the connection with a capacity equal to the PCR. But since VBR services do not always operate at their peak

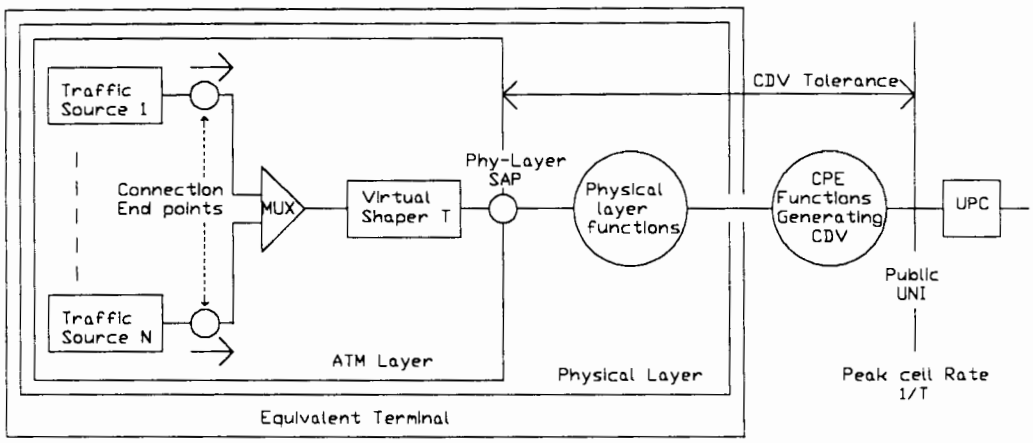



Figure 3-1 Reference Model

rate, this means that the connection would be under-utilised most of the time, and would be inefficient.

Of the four parameters specified above only three will be used as inputs to the proposed controller, namely the PCR, SCR and BT. The controller will thus need modules to monitor these values. Even though these three parameters are used with VBR traffic, the controller can also be used to monitor CBR. This is done by redefining a CBR source as a VBR source with its SCR set equal to its PCR and its BT set to infinity. The controller can thus monitor both VBR and CBR traffic because basically CBR is just a "subset" of VBR. Combining these two traffic classes also help to simplify the design of the controller.

## 3.2 The Control Action

The next thing to consider is what action should be taken when traffic is non-conforming. Two of the possibilities are to either drop all non-conforming cells, or tag the cells whose degree of non-conformity is not too excessive. Other possibilities include closing the connection, applying traffic shaping and providing feedback to the source. These other possibilities will however not be discussed here.

How does one determine when non-conformity is too excessive, or how does one determine what the threshold between tagging cells and dropping cells is? There is no straight forward answer to this question. If too many non-conforming cells are tagged and let through to the network, then it means that the responsibility of dealing with those non-conforming cells has simply been passed on to the network when it should have been handled at the UNI. These cells inevitably use network resources and put additional strain on the network. Even though the cells have been tagged, indicating that they may be dropped if congestion occurs in the network, the fact remains that those cells still have to be processed, even if it is only to drop them. On the other hand if too few cells are tagged, it means that traffic that could have been handled by the network is discarded prematurely. A careful balance should thus be maintained between maximising the amount of traffic that is allowed through to the network, and minimising the impact that non-conforming traffic will have.

One possible answer to this question is to use the discretion of an expert who is familiar with this problem to make the control decision. If it could be possible to extract that decision making processes from the expert and apply it in the controller, then the problem of determining the threshold point would be solved. Fuzzy Logic allows one to do this [ 12 ].

Now that there is a way of deciding between tagging and dropping cells, one can include cell tagging as an option in the control action of the controller.

The control action will thus be to either tag or drop non-conforming cells, and let conforming cells through to the network unmodified.

## 3.3 The Initial Fuzzy Logic Congestion Controller

While doing the initial research on the topic of preventive congestion control in ATM networks at the start of the thesis, the author found that there were many papers written on congestion control using the more traditional methods, but could not find any describing the use of fuzzy logic as part of their design. After having read through several papers describing complex mathematics and queuing theory, and seeing the amount of statistics involved when trying to produce such a controller, the author found it odd that no-one had yet (as far as he knew) used fuzzy logic to control this rather complex process.

Thinking back on his basic undergraduate course notes the author knew that fuzzy logic should be able to control such a poorly defined process as preventive congestion control in ATM. Using this basic idea and some basic knowledge of classical control theory, the author set out to design such a controller. This initial design is shown in Figure 3-1 and was presented in [ 14 ].

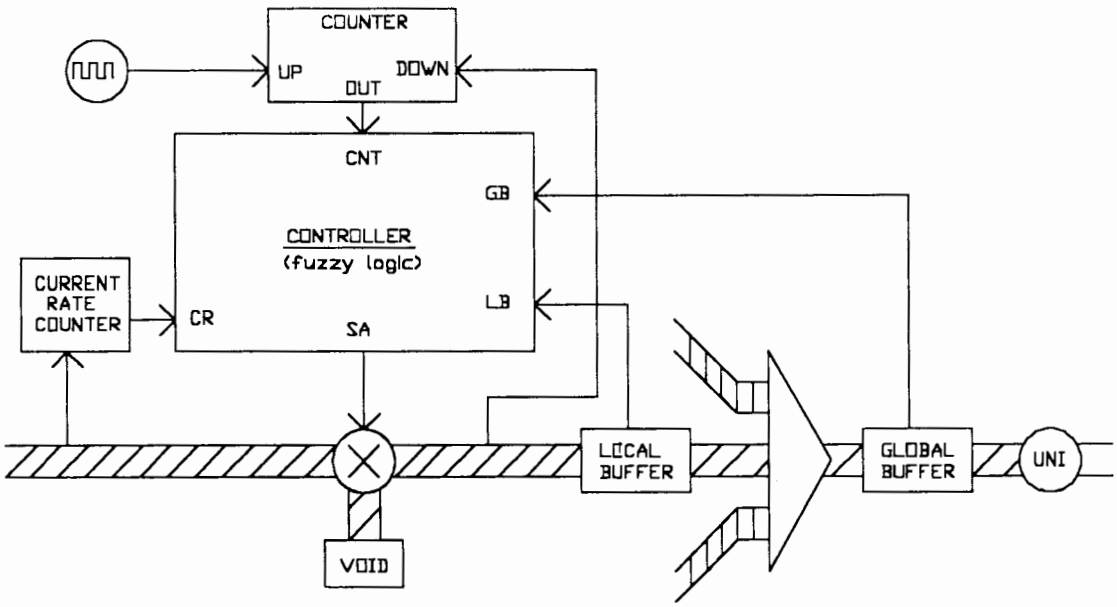



Figure 3-1 The Initial Fuzzy Logic Congestion Controller

Quoting from [ 14 ], Figure 3-1 "shows a proposal on how to control the flow of ATM cells before they enter the UNI. The heart of the system is a Fuzzy Logic Controller

(FLC) that analyses the state of the system and makes a decision as to what control action to take." The Fuzzy Logic Controller's support system consists of the following:

1. a current bit rate counter,
2. an up-down counter,
3. two cell buffers,
4. and a switch.

Depending on what state the system is in, the FLC either lets the cells through the switch, tags the cells before allowing them through, or drops the cells. This method of putting the switch at the front end of the system is used to allow the controller to discard any non-conforming cells before they enter the network and use up valuable system resources. Please refer to [ 14 ] for a more detailed description.

Before the paper was submitted, the author finally found three other papers [ 11 ],[ 15 ] and [ 16 ], that also used fuzzy logic as part of their design. He was rather surprised to find that his initial design had some features that were common to the design that Ndousse presented in [ 9 ], and to the design that Jensen presented in [ 15 ].

Several comparisons can be made between Ndousse's design and the initial design shown in Figure 3-1. Ndousse presented the idea of using a fuzzy logic implementation of the virtual leaky bucket mechanism [ 17 ] to control voice cells in ATM networks. The up-down counter was included in Figure 3-1 to try and incorporate the idea of a token leaky bucket into the design. Ndousse had a Quality of Service (QoS) buffer that provided feedback on the state of the connection to the fuzzy logic controller; Figure 3-1 had a local buffer for the same reason. Ndousse's controller, however, only considered the state of one connection, while Figure 3-1 shows an additional buffer that is used to monitor the combined effect of all the connections before they enter the ATM pipe.

Jensen presented the use of fuzzy logic for B-ISDN network management. His fuzzy logic controller monitors the input as well as the output links. Jensen's idea of monitoring the output link corresponded with the author's idea for including his output buffer in Figure 3-1. Now, looking back at the very first design, the author feels that it

was not necessary to use a buffer at all; Jensen's way of simply measuring the bandwidth is more elegant and much simpler indeed.

In the end it was decided not to implement this initial design. There was one basic design problem that proved particularly difficult to solve, namely, what size to make the buffers. Using a buffer that was too large would mean wasting resources, using a buffer that was too small would degrade the performance of the controller, and sharing buffer space between different controllers would simply complicate the design. Intuitively it was felt that each connection would need a different buffer size.

The author did not specify the size of the buffers used in [ 14 ], and found that Ndousse and Jensen also did not specify theirs. This seems to be a fundamental design problem in their, as well the author's, design. Cheng also mentions in [ 11 ] that it is difficult to decide on the size of the buffer if one does not have complete statistics of the input traffic. One finds that the size of the buffers is generally determined by trial and error or that some heuristic search algorithm is used to find the correct value.

It was felt that the initial design was not good enough because of this design problem. Another was required to take its place. The initial design was not simulated. It was felt that too many simulations would have to be run to determine firstly the buffer sizes and then tune the fuzzy sets. This would have been too time consuming. The author thus decided to abandon this design and proceed with the next one.

The second design would have to operate without buffers in order to overcome the failure of the first design, and still be as simple as possible.

## 3.4 The Final Fuzzy Logic Congestion Controller

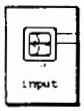
The design can now be finalised by considering the parameters that were chosen to characterise the traffic as well as the desired control actions.

Separate modules were used to handle the different aspects of the design. A modular approach was followed so that the inter-relation between the traffic parameters, the control actions and the design functions of the controller could be clearly shown. This would also help to keep things as simple as possible.

### 3.4.1 The Design Modules

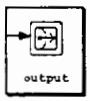
Several modules were required, their operation and reason(s) for inclusion are discussed below.

#### *3.4.1.1 The Input Module*



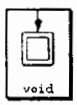

The Input Module accepts cells coming into the Fuzzy Logic Congestion Controller and passes those cells on to the Switch Module. It also informs the Current Rate Counter when a new cell arrives.

#### *3.4.1.2 The Output Module*



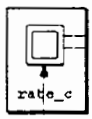

The Output Module accepts cells from the Switch Module and passes them on to the UNI.

#### *3.4.1.3 The Void Module*




The Void Module accepts cells from the Switch Module and destroys them.

#### *3.4.1.4 The Rate Counter*




The Rate Counter is used to monitor the rate at which cells enter the Input module. It contains two counters that is used to determine the average and the current cell rate.

The first counter is incremented by one each time an incoming cell is detected, i.e. it counts the number of cells detected since the start of the connection. Averaging this

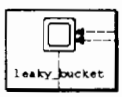
counter value over the current duration of the connection, provides the Fuzzy Logic Controller with the average cell rate. This value is stored as the Average Cell Rate (ACR) variable.

The Average Cell Rate is used to help monitor the sustainable cell rate traffic parameter. The SCR is defined over the entire duration of the connection and is constant. It determines what the average cell rate will be more or less when the connection is closed. But since one can't wait until the connection terminates before starting to see if the traffic has kept within its traffic contract, the Average Cell Rate is used to determine what the average cell rate is while the connection is still open. Ideally the average cell rate will be equal to the sustainable cell rate when the connection terminates.

The Fuzzy Logic Controller is a sampled system. The second counter thus counts how many cells arrived since the last time the Fuzzy Logic Controller requested information from the Rate Counter. Averaging this value over the time that has elapsed since the last request, provides the current cell rate of the connection. This value is stored as the Current Cell Rate (CCR) variable.

These two variables are presented to the Fuzzy Logic Controller on request[5], allowing the controller to monitor two of the values needed to evaluate the state of the connection.

*3.4.1.5 The Up-Down Counter*

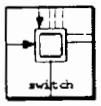



The Up-Down Counter is added to monitor the rate at which cells are allowed through the Switch Module. This counter has a reference clock that periodically tells it to increment by one unit. The period is set to the same value as the negotiated sustainable cell rate. Every time a cell is allowed through the switch, the counter is decremented by one unit. The counter value is sent to the Fuzzy Logic Controller when requested.

The Up-Down counter was included to help monitor the burst duration, i.e. the burst threshold (BT) parameter. It will be discussed in more detail in section 3.5.3.

[5] i.e. Data is only sent when the Fuzzy Logic Controller requires it.

*3.4.1.6 The Switch*




The Switch is the module that executes the control action of the Fuzzy Logic Controller. Depending on the command received from the Fuzzy Logic Controller, it performs one of the following operations on a cell received from the Input module:

1. routes the cell to the Output module unmodified,
2. tags the cell before routing it to the Output module, or
3. routes the cell to the Void module to be destroyed.

When a cell is routed to the Output module, the Switch module informs the Up-Down Counter to decrement itself by one unit.

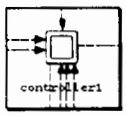
Note that the Switch module on its own has no intelligence, it depends on the Fuzzy Logic Controller for instructions. What it *does* is keep on executing the last instruction on the incoming cells until it receives a new instruction. This functionality is needed so that if the Fuzzy Logic Engine is too slow at evaluating the current status of the connection, then cells need not be stored until the evaluation is complete. In this manner the need for cell buffers before the switch is eliminated. Conceptually one can think of the Switch as assuming that the state of the connection is the same as it was the last time the Fuzzy Logic Controller sent it an instruction, and therefore continues to execute the last instruction that it received. The drawback of this method is that a sudden burst of non-compliant cells may be let through before the Fuzzy Logic Controller determines that the non-compliant burst of cells *is* actually non-compliant. This is unpreventable in a sampled system like this. One therefore has to make sure that the Fuzzy Logic Engine is fast enough at evaluating the state of the connection to handle the rate at which cells are entering the controller.

A cell is tagged to indicate to the network that it may be dropped if needed to prevent congestion further on in the network[6], and indicates a cell of low priority.

[6] This feature could also be exploited if the Fuzzy Logic Congestion Controller was to be placed at the input of switches *inside* the network. A slight modification of the Switch module would cause a tagged cell coming into the switch to be dropped if the Fuzzy Logic Controller determines that the cells need to be tagged. In this manner non-conforming cells that have been tagged already, will not continue to propagate through to the network during a period of congestion.

Three counters are included in the Switch module to keep track of how many cells were left unmodified, how many were tagged, and how many were dropped. These counter values are passed to the Fuzzy Logic Controller on request, and are only included for information gathering purposes.

*3.4.1.7 The Fuzzy Logic Controller*




The Fuzzy Logic Controller is the heart of the Fuzzy Logic Congestion Controller and as such is the most complex module in the system:

1. It contains the Fuzzy Logic Engine that evaluates the state of the connection,
2. collects the negotiated traffic parameters when the connection is set up, and
3. serves as interface to higher functional layers that manage the Fuzzy Logic Congestion Controller.

Several things happen when the connection is established:

1. The counters in the various modules mentioned above are reset to zero.
2. If a VBR service was requested then the Fuzzy Logic Controller collects the negotiated peak cell rate, the negotiated sustainable cell rate and the burst duration at peak rate of the connection. If a CBR service was requested then it only collects the peak cell rate. The CBR service is then treated as a VBR service as was described when discussing the traffic parameters in section 3.1.
3. The Switch module is instructed to let all cells through unmodified.

The Fuzzy Logic Controller starts to monitor and evaluate the state of the connection once the initialisation has completed. The input variables to the controller are sampled at the start of each evaluation cycle, fuzzyfied and passed to the Fuzzy Logic Engine where the rule base is applied and the control action generated. The control action is then converted and sent to the Switch module as an instruction. The conversion process will be discussed in section 3.5.4.

### 3.4.2 The complete design

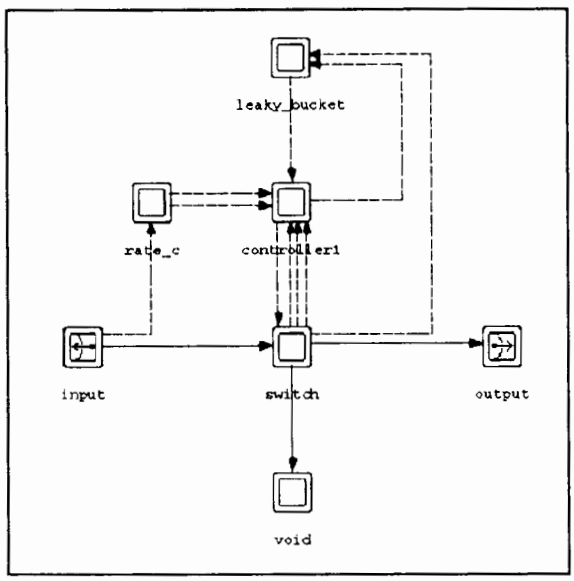



Figure 3-1 The Fuzzy Logic Congestion Controller

Figure 3-1 shows the design chosen to control the flow of ATM cells into the UNI. Solid lines show the path a cell can possibly follow, while dashed lines show the paths of internal communication between the modules. Cells enter on the left hand side at the Input Module and move to the Switch Module. Depending on whether cells are conforming or not, they either exit through the Output module or are destroyed by being routed into the Void module. The rate counter monitors the incoming average and current cell rate, while the leaky bucket helps to determine the burst size.

No distinction is made between cells containing data and resource management cells. Either can be dropped if deemed necessary. This is done to keep the design as general as possible. If such functionality was needed then it would not be difficult to build it into the Switch Module.

Little has so far been said about the fuzzy logic engine that forms the heart of the controller. When designing any Fuzzy Logic Controller there are two basic processes that one needs to go through to make sure that the fuzzy logic engine can function properly. The first is defining the fuzzy sets so that variables can be fuzzyfied and defuzzyfied, and the second is deciding what rules to use in the decision making process. Both are discussed in the next section.

## 3.5 The Fuzzyfication and Defuzzyfication Process

The Fuzzyfication process enables the real valued variables to be expressed linguistically. This involves converting them into a form that the Fuzzy Logic Engine can handle[7], while the defuzzyfication process converts the output variable back into a real valued variable.

During the planning stages of the design it was decided that the Fuzzy Logic Controller should be kept as simple as possible. This was done to shorten the time the Fuzzy Logic Engine takes to evaluate the state of the connection. The shorter this evaluation time can be made, the faster the Fuzzy Logic Congestion Controller will be able to respond to the incoming cells, and therefore the better its performance will be.

The duration of an evaluation is directly proportional to the number of rules that need to be evaluated, while the number of *possible* rules is again heavily influenced by the number of fuzzy sets assigned to each fuzzy variable. Thus, in order to minimise the evaluation time, each fuzzy variable should have as few fuzzy sets as possible.

The controller has three input variables. How does one decide how many sets to assign to each one? This depends mainly on two things:

1. how many descriptive terms should one use to describe a variable, and
2. how complex does one want the rule base to be.

The first point determines how sensitive the controller is to a change in an input variable, while the second point determines how many possible rules will be in the rule base.

One could, for example, describe the average rate as being either HIGH or LOW. This translates to two fuzzy sets for that fuzzy variable. The rule base would thus also have to consider only two states of the average rate. In general, the more sets per variable the more sensitive the system and the smoother the control action will be. Having only two

[7] Please refer to Appendix A for a short primer on Fuzzy Logic.

sets could mean that the controller might not have enough rules to be sensitive enough when considering this variable.

Keeping this in mind, one could choose to use more descriptive states, i.e. sets, for the average rate. One could for example choose to describe it as VERY SLOW, SLOW, FINE, FAST, VERY FAST and CRITICAL. This translates to six fuzzy sets for this variable and therefore the rule-base also has to consider its six possible states. This means a smoother control action, but also that a lot of rules will need to be evaluated in the process, which might slow the controller down.

If one now decides to assign the same sets to describe the current rate, there could be as many as thirty-six rules in the rule-base. When adding the up-down counter variable as well, one ends up with N x 36 possible rules if there are N sets assigned to the up-down variable. Here one sees that the maximum number of rules in the rule-base is equal to the arithmetic product of the number of sets in each variable for all the input variables. In this case that would be 6 x 6 x N.

One should also note that not all variables have to have the same number of sets. In general, the more sensitive a design is to the change in the state of a variable, the more sets are assigned to that variable, and vice versa.

After carefully considering these factors a maximum of four fuzzy sets were chosen per variable. This low value would help to keep the rule base down to a manageable size - a maximum of sixty-four rules to be exact. Again, since this is an expert system, it is up to the expert or designer of the system to decide how many rules need to be in the rule base. In general the rule base is kept as full as possible in order to cover as many of the possible states of the system as possible.

The shapes of the fuzzy sets are also kept as simple as possible. Only one trapezium, triangle or singleton is used when defining the shape of a fuzzy set. These simple shapes were chosen to simplify the mathematics involved when the Fuzzy Logic Engine applies the inference process.

Each fuzzy variable and the sets that define their possible states will now be discussed in more detail.

### 3.5.1 The Average Cell Rate

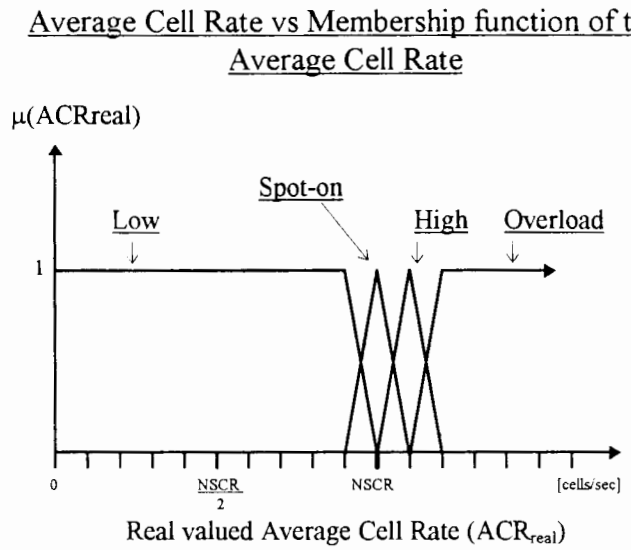



Figure 3-1 The fuzzy sets defining the Average Cell Rate.

Figure 3-1 shows how the Average Cell Rate is described by the shape, position and description of its four sets. The horizontal axis represents the real valued[8] Average Cell Rate ($ACR_{real}$), while the vertical axis represents the membership function of $ACR_{real}$. NSCR denotes the Negotiated Sustainable Cell Rate.

Figure 3-1 compares the $ACR_{real}$ with the NSCR so that the relationship between the two values can be described in words. This is done in the following manner:

- if the $ACR_{real}$ is less (or lower) than the NSCR then the $ACR_{real}$ is said to be "Low",
- if the $ACR_{real}$ is almost equal to NSCR then the $ACR_{real}$ is said to be "Spot-on",
- if the $ACR_{real}$ is slightly bigger than NSCR then the $ACR_{real}$ is said to be "High", and lastly
- if the $ACR_{real}$ is much bigger than NSCR then the $ACR_{real}$ is said to be in a state of "Overload."

Mathematically the sets are defined as follows:

[8] i.e. not fuzzyfied

Let a triangular function $f(x; x_0, a_0, a_1)$ be defined as:

$$f(x;x_0,a_0,a_1) = \begin{cases} \dfrac{x - x_0}{a_0} + 1 & \text{for } x_0 - a_0 < x < x_0,\ a_0 \neq 0 \\ \dfrac{x_0 - x}{a_1} + 1 & \text{for } x_0 < x < x_0 + a_1,\ a_1 \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

and a trapezoidal function $g(x; x_0, x_1, a_0, a_1)$ be defined as:

$$g(x;x_0,x_1,a_0,a_1) = \begin{cases} \dfrac{x - x_0}{a_0} + 1 & \text{for } x_0 - a_0 < x \leq x_0,\ a_0 \neq 0 \\ 1 & \text{for } x_0 < x < x_1 \\ \dfrac{x_1 - x}{a_1} + 1 & \text{for } x_1 < x \leq x_1 + a_1,\ a_1 \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $x_0$ in f( ) is the centre of the triangular function; $x_0$ and $x_1$ in g( ) are the left and right edge of the trapezoidal function; and $a_0$ and $a_1$ are the left and right width (respectively) of the triangular or the trapezoidal function.

Let $\mu_{LOW}(q)$, $\mu_{SPOT\text{-}ON}(q)$, $\mu_{HIGH}(q)$, and $\mu_{OVERLOAD}(q)$ denote the membership functions for sets LOW, SPOT-ON, HIGH, and OVERLOAD in ACR(q) respectively, and let $\mu_{LOW}(q)$, $\mu_{SPOT\text{-}ON}(q)$, $\mu_{HIGH}(q)$, and $\mu_{OVERLOAD}(q)$ be defined as

$$\mu_{Low}(q) = g(q, 0, NSCR * 0.9, 0, NSCR * 0.1) \tag{4}$$

$$\mu_{Spot-on}(q) = f(NSCR, NSCR * 0.1, NSCR * 0.1) \tag{5}$$

$$\mu_{High}(q) = f(NSCR * 1.1, NSCR * 0.1, NSCR * .01) \tag{6}$$

$$\mu_{Overload}(q) = g(NSCR * 1.1, \infty, NSCR * 0.1, 0). \tag{7}$$

Equations (1) to (6) allow us to describe the Average Cell Rate linguistically, and once this step is done we can "take a step back" from the mathematics involved in the system.

Now we can say "If the ACR is HIGH and ..., then ... ." This makes the design of the new controller much simpler[9].

Considering the decision to use a maximum of four sets per fuzzy variable, the choice of four sets to represent the ACR means that the ACR is an important variable and should be carefully considered when setting up the rule base.

### 3.5.2 The Current Cell Rate

Figure 3-1 can also be used to describe the Current Cell Rate (CCR) but with the NSCR replaced by the Negotiated Peak Cell Rate (NPCR). This change is shown in Figure 3-2.

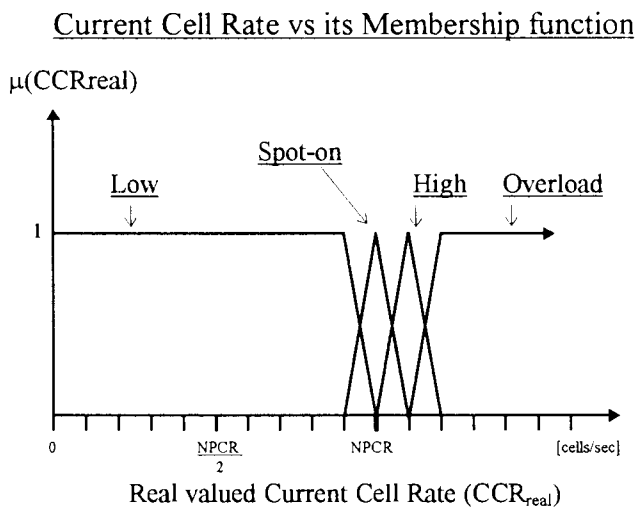



Figure 3-2 The fuzzy sets defining the Current Cell Rate.

Comparing Figure 3-1 and Figure 3-2 one sees that the linguistic and mathematical description of the CCR is similar to that of the ACR in section 3.5.1 (remembering to

[9] For example, the author's thesis supervisor gave this excellent example of how a train driver manages to stop a train without the train stopping before or after the station. Here the driver is presented with a very complex system, that of bringing a large moving machine to a halt at (or very close to) a particular point without damaging the train or its passengers. A train is a very complex machine, and to take a mathematical approach of designing a control system for stopping it, would need to involve taking into account all the objects attached to or inside it, the interaction of the train with its environment, the amount of friction the wheels have on the tracks, how heavy the train is, its speed, the state of the tracks (wet or dry), etc., etc., etc. But, the train driver does not need all these variables in order to decide when and how hard to apply the brakes. What the train driver would typically do is say "If the train is going very fast and the tracks are wet and the train is fairly full then brake early and not so hard." Here, effectively, the train driver has "taken a step back" from the complexity of the train (or system) by describing the state of the system in words and applying the rules he/she gained through experience in order to achieve the task of stopping the train.

replace NSCR with NPCR) and is thus not repeated here. Note that CCR also has four sets and is considered an important variable.

### 3.5.3 The Up-Down Counter Value

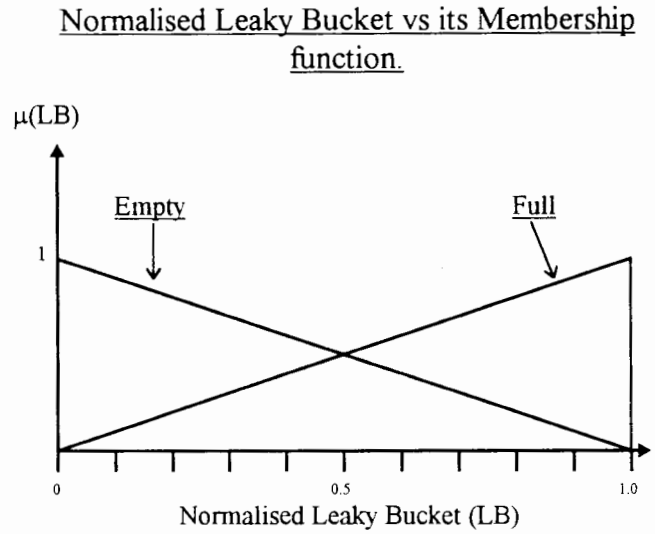



Figure 3-3 The fuzzy sets defining the Leaky Bucket.

Figure 3-3 describes the Up-Down counter variable. At closer inspection one sees that "Up-Down counter" is not mentioned in Figure 3-3, but instead a variable called "Leaky Bucket" is used instead. This is because "Leaky Bucket" better describes what the variable will be used for. Henceforth the Up-Down counter variable will be referred to as the Leaky Bucket (LB) variable.

The Up-Down counter variable was renamed for the following reason. During the design of the controller the author found that control schemes based on rate control are popular in the design of traffic controllers because of the simplicity of their design and the ease with which they can be implemented. Figure 3-4 shows one such control scheme where cells arriving at a token leaky bucket are let through if there are tokens available in the bucket, and are dropped if the bucket is empty[10].

[10] Each cell "takes" a token out of the bucket as it is allowed through. Tokens are placed back into the bucket at a predetermined constant rate until the bucket is full.

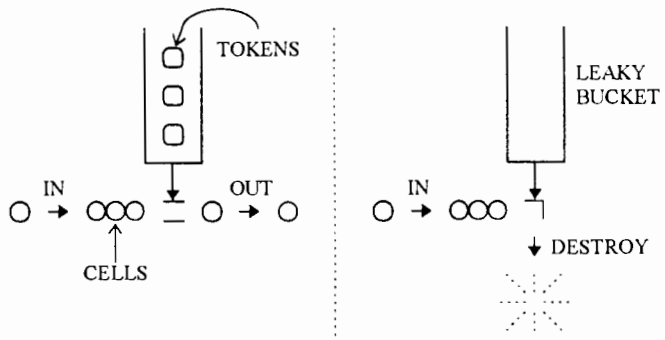



Figure 3-4 Simple Credit based control using a token Leaky Bucket.

Le Boudec however summarised in [ 13 ] that rate control (of VBR traffic) using such simple schemes (such as the token bucket, leaky bucket and windowing algorithms) is not effective enough on their own, and that "more sophisticated rate control schemes" should be used. Following Le Boudec's suggestion the author decided to incorporate a simple credit based control scheme using a token leaky bucket into the Fuzzy Logic Controller. In this design however the leaky bucket is not used to directly control the flow of cells through the controller, but merely "suggests" to the Fuzzy Logic Controller what control action to take.

Here the leaky bucket will be used to determine when to start tagging or dropping cells. In cases where the average cell rate is marginally more than the rate agreed upon, the contents of the bucket will slowly drop down and will indirectly reflect the "history" of the source. "History" in the sense that if the bucket is low then it means that too many cells have been allowed through by controller in the recent past, and that some control action should be taken.

The state of the bucket is then considered in the rule base to determine what control action to take. It is the job of the Fuzzy Logic Controller to decide whether or not to allow cells through the Switch module depending on the state of its other input variables. Cases can thus arise where a normal credit based controller would throttle[11] traffic while the Fuzzy Logic Controller would let traffic through. One such case would be if the Fuzzy Logic Controller determines that the average cell rate is below the negotiated rate and that a short burst of incoming cells would not yet violate the traffic

[11] Prevent cells from passing through the Switch module.

contract. The Up-Down counter and the fuzzy logic in the Fuzzy Logic Controller combine to form this leaky bucket.

Figure 3-3 shows that the Leaky Bucket can be described as being Full and Empty. Only two sets are used to describe this fuzzy variable, which shows that the system is not as sensitive to a change in its state as it is to changes happening in the ACR and CCR. Stated in another way, the average cell rate bears more weight in the decision making process than a sudden burst of traffic.

Figure 3-3 also shows that LB is normalised; the normalisation value chosen as the size of the leaky bucket. The size of the leaky bucket depends on the NPCR and the maximum burst duration at the NPCR, and is set to the product of these two values. This means that, in the case of a *simple* leaky bucket, if the bucket was *full* and a sudden burst of cells arrived at the NPCR, then cells would only be allowed through for the maximum burst duration. Thereafter cells would be dropped because the bucket would be empty. Obviously, fewer cells would be allowed through if the bucket was not full to start off with. In the case of the "fuzzy leaky bucket" the bucket can for example be both sixty percent Empty and forty percent Full at the same time. Thus, the decision to drop cells if the bucket is Empty can not be applied to the fuzzy leaky bucket in the same way as for the simple leaky bucket because a degree of "Fullness" will always apply as well. This is where the Fuzzy Logic Engine takes the decision making process away from the leaky bucket.

The Leaky Bucket variable is defined mathematically as follows:

Let $\mu_{EMPTY}(q)$ and $\mu_{FULL}(q)$ denote the membership functions for sets EMPTY and FULL in LB(q) respectively, and let $\mu_{EMPTY}(q)$ and $\mu_{FULL}(q)$ be defined as

$$\mu_{Empty}(q) = f(1,0,1) \tag{8}$$

$$\mu_{Full}(q) = f(1,1,0)\,. \tag{9}$$

### 3.5.4 The Switch Action

The Switch Action (SA) is the last fuzzy variable that needs to be defined. It is the output variable of the Fuzzy Logic Controller and as such represents the control decision thereof. Each connection has its own SA and is not dependant on the traffic

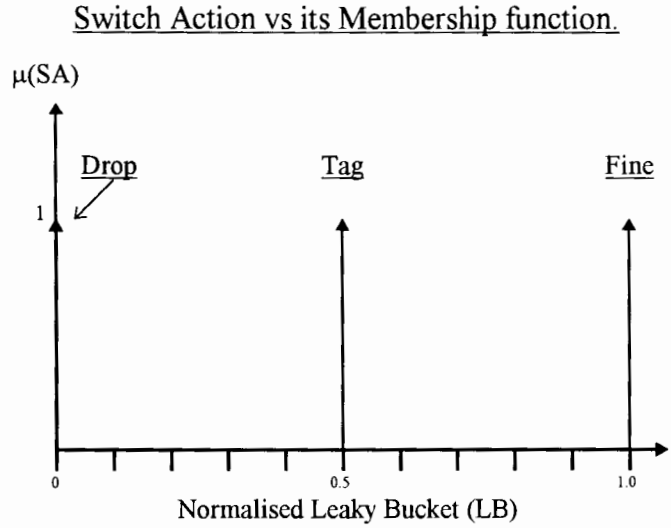



Figure 3-8 The sets describing the Switch Action.

generated in other connections in the UNI. Figure 3-8 describes the SA graphically.

Figure 3-8 shows that the SA can be described as Drop, Tag, and Fine, and defines three possible control actions that the Fuzzy Logic Engine can come up with: namely to drop the cell, tag the cell, or leave the cell as is because it is within the traffic contract.

Only three sets are chosen because only three different actions are possible. The sets, in this case, are defined with singletons because the SA is an output variable, and are defined mathematically as follows:

Let a function s known as a singleton be defined as

$$s(x;x_0) = \begin{cases} 1 & for\ x=x_0 \\ 0 & otherwise \end{cases} \tag{10}$$

Let $\mu_{DROP}(q)$, $\mu_{TAG}(q)$ and $\mu_{FINE}(q)$ denote the membership functions for sets DROP , TAG and FINE in SA(q) respectively, and let $\mu_{DROP}(q)$, $\mu_{TAG}(q)$ and $\mu_{FINE}(q)$ be defined as

$$\mu_{Drop}(q) = s(0) \tag{11}$$

$$\mu_{Tag}(q) = s(0.5) \tag{12}$$

$$\mu_{Prune}(q) = s(1.0)\,. \tag{13}$$

Figure 3-5 is used in the *defuzzyfication* process because SA is an *output* variable.

When the defuzzyfication process is complete one is left with a real valued variable that represents the control decision of the Fuzzy Logic Controller. But the Switch Module can only execute one of three instructions, let cells through, tag the cells or drop them. One thus needs to map the real valued variable over to one of the three instructions that the Switch Module can execute.

Figure 3-5 is used to re-map the decision variable. The traffic is accepted as be conforming if it is above the tag level, i.e. bigger than 0.5. The choice of when to drop or tag cells determines how strict the controller is at policing non-conforming traffic. After testing several cut-off values, a value of 0.2 was chosen as the threshold. Now if the decision variable is bigger than 0.2 and less than or equal to 0.5, then cells are tagged. If less than 0.2 then cells are dropped.

This concludes the fuzzification and defuzzyfication processes. Now that the Fuzzy Logic Engine has the linguistic equivalent of its input variables and is capable of converting its output linguistic variable to one of three instructions, one needs to consider the rule base that is used to produce the output variable.

### 3.6 Setting up the Rule Base

Fuzzy Logic is an Expert System that uses existing knowledge about a system or problem and applies that knowledge to evaluate or solve a problem concerning that system. The idea of having stored knowledge of the system is central to the operation of such a system.

One method of obtaining this knowledge is by extracting the information from an expert who has experience with the system in question and storing it in the Knowledge Base of the system. On the other hand, if an expert is not available for the problem at hand, then another method used is to study the system and learn its behaviour in order to *become* the expert of the system. In both cases it is the "quality" of the knowledge that is extracted that determines how well the Expert System will perform in the end. "Garbage in, garbage out," is an important concept to keep in mind when deciding what to put into the Knowledge Base.

There was no expert knowledge available about the Fuzzy Logic Controller presented in this chapter when the design started, and therefore the second of the two methods mentioned above was chosen to help obtain the expert knowledge needed for the controller. The process of gaining this knowledge was fairly abstract and was obtained by means of a combination of carefully studying the layout of the design, applying common sense in what the controller should and should not do, and developing a "gut feel" for the operation of the controller. This might not seem like a very "scientific method" for getting such information, but this is in fact how most humans learn anything, and it is the results of this learning process that the Fuzzy Logic Expert System takes advantage of.

The knowledge obtained in this manner is stored in the knowledge base of the system by means of rules that describe the action required when certain combinations of the input variables occur. A rule base is built up by defining a set of rules for all the possible states of the input variables. Table 1 and Table 2 summarise the rules chosen for the Fuzzy Logic Congestion Controller.

| CCR \ ACR | LOW | SPOT-ON | HIGH | OVERLOAD |
|---|---|---|---|---|
| LOW | FINE | FINE | TAG | DROP |
| SPOT-ON | FINE | FINE | TAG | DROP |
| HIGH | FINE | **TAG** | DROP | DROP |
| OVERLOAD | TAG | DROP | DROP | DROP |

Table 1 Rule Base: Current Cell Rate vs. Average Cell Rate, Bucket is Full.

| CCR \ ACR | LOW | SPOT-ON | HIGH | OVERLOAD |
|---|---|---|---|---|
| LOW | FINE | FINE | DROP | DROP |
| SPOT-ON | FINE | TAG | DROP | DROP |
| HIGH | TAG | DROP | DROP | DROP |
| OVERLOAD | DROP | DROP | DROP | DROP |

Table 2 Rule Base: Current Cell Rate vs. Average Cell Rate, Bucket is Empty.

Table 1 shows the rule base when the fuzzy leaky bucket is FULL, while Table 2 applies when the fuzzy leaky bucket is EMPTY. The rows represent the CCR while the columns represent the ACR.

The bold face entry in Table 1 is a rule in the rule base and is read as follows:

*"If ACR is Spot-on and CCR is High and the Bucket is Full, then set SA to TAG."*

The other rules should be read in a similar fashion.

Looking at the two tables one sees that Table 2 is more strict when deciding what the SA should be. For example, if the ACR is High and the CCR is Spot-on then Table 1 shows that the cell should be tagged, while Table 2 shows that it should be dropped. This agrees with the idea that cells should be dropped when the fuzzy leaky bucket is empty.

## 3.7 In Summary

The entire process of how the Fuzzy Logic Controller generates its output can quickly be described as follows:

> Using Figure 3-1, Figure 3-2 and Figure 3-3 the crisp inputs from the system are fuzzyfied. Table 1 and Table 2, the Max-Min compositional rule of inference, and the Centroid method is then used to produce the crisp SA, the control decision, which ranges from 0 to 1. If the SA is bigger than 0.5 then cells are considered to be conforming and should be let through unmodified; if less than 0.2 they should be dropped; any other value results in tagged cells. The control decision (FINE, TAG, or DROP) is then sent to the Switch module for implementation.

This concludes the design of the Fuzzy Logic Congestion Controller. The next chapter presents the simulation of this controller.

# 4. SIMULATION OF THE SECOND FUZZY LOGIC CONTROLLER

Chapter 3 presented the two initial designs of the Fuzzy Logic Congestion Controller. The first design was not simulated due to the difficulty of determining what size to make the buffers in the controller, and was abandoned as a result. This design problem lead the author to the second design that overcame the buffer problem by removing the buffers from the controller. A simple token leaky bucket was inserted to provide the necessary feedback instead. This chapter presents the simulation results, and evaluation of the second design.

The fuzzy sets and rules presented in chapter 3 were converted into a format that Fuz, the fuzzy logic inference engine, could handle[12], and was simulated using OPNET®. At that stage the shapes of the fuzzy sets were set to the initial values specified, and as such were un-optimised. The optimisation took place during the various simulation runs by making small changes in the width of the sets, and seeing how it affected the performance of the controller.

Each simulation was run for a predetermined duration, during which time OPNET® wrote the desired statistics of the simulation to disk. No statistics were presented to the user while the simulation was in progress. This meant that the design would have to be analysed by the results that were generated during the run.

During the initial stages of the simulation it was felt that a simple traffic source was needed to test the controller. It would be of no use, for example, to have a VBR source with so much variation that it looked in the simulations results as if a random source had been used instead. Being able to understand and interpret the results was of utmost importance in evaluating the performance of the controller. A simple source would make it much simpler to verify that the controller was working the way it was designed to. For these reasons a simple sinusoidally varying VBR source was chosen. The user would be able to specify a base (constant) rate and also specify by how much that base rate would vary. It is the sinusoidal variation in the base rate that makes the source vary its rate sinusoidally between a positive minimum and maximum rate.

[12] Please refer to Appendix B for a description of Fuz and the format of its input files.

Three test cases will be considered in this chapter:

1. The source is always conforming,
2. The source is conforming most of the time, and
3. The source is non-conforming.

Each test case is presented with the simulation results obtained from OPNET®, and is then evaluated in order to determine the performance of the controller. After evaluating all three cases a decision is made whether to accept the controller as is or whether further design changes are needed.

The results shown are those obtained after the fuzzy sets have been optimised. It was found that the optimised fuzzy sets did not differ much from the initial sets, and as such will only be presented for comparison with the original.

## 4.1 The Simulation Results

Several statistics were gathered during each simulation and were logically placed into three groups to simplify the performance evaluation of the controller. The three groups consisted of the following:

1. The inputs and output of the controller,
2. the number of cells that were fine, tagged, and dropped by the Fuzzy Logic Controller, and
3. the number of cells that were determined by the GCRA to be conforming and non-conforming.

The first group of statistics were normalised in order to be presented together on the same graph. In this manner the Current Rate was normalised to the negotiated peak rate and labelled as the RATE-COUNTER; the Average Rate was normalised to the negotiated average rate and labelled as the TIME-AVERAGE; while the current contents of the bucket was normalised to the size of the bucket and labelled as BUCKET. The crisp output value of the fuzzy logic engine was chosen as the switch action statistic and was labelled as SA. This value ranged from zero to one and thus did not require normalisation.

The second group of statistics were labelled as FINE-PACKETS, TAG-PACKETS, and DROP-PACKETS. These values are presented as the number of cells that were affected by the three possible control actions of the controller. They were not normalised as such action would be meaningless.

The GCRA statistics are simply labelled Conforming and Non-Conforming and are used to compare the results obtained from the Fuzzy Logic Controller.

### 4.1.1 Case 1 - The source is always conforming.

The first case considers a simple VBR source that is sending at a steady rate below the negotiated average cell rate, i.e. it performs like a CBR source. Figure 4-1 shows that the RATE-COUNTER and TIME-AVERAGE are both less than 1, i.e. less than the negotiated peak and average rates respectively, and that the BUCKET stays full. This means that the

control action of the controller should allow all cells through, and is shown by SA staying close to 1. The controller is thus performing as expected.

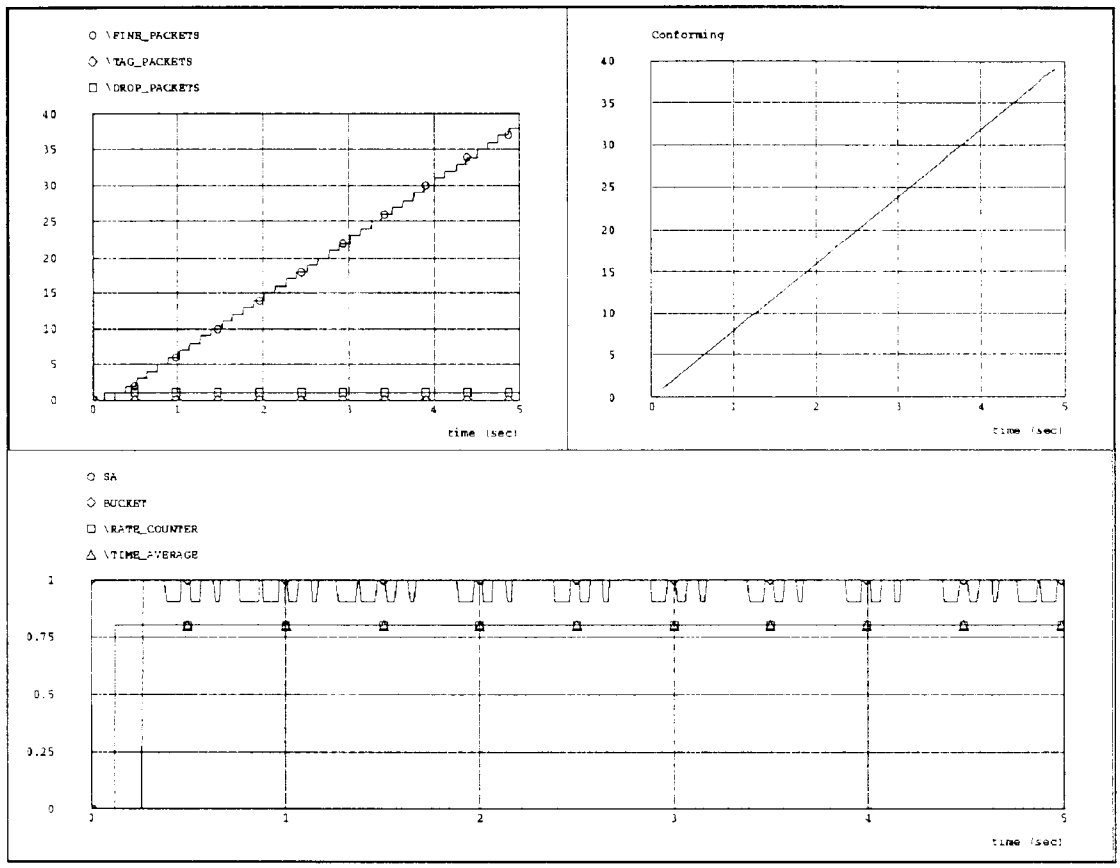



Figure 4-1 Case 1 - A conforming source.

### 4.1.2 Case 2 - The source is conforming most of the time.

The sinusoidally varying VBR source was chosen for this case. It ranged from a minimum rate of 35% to a maximum of 87% of the negotiated peak rate. This can be seen in Figure 4-2. The negotiated average rate was set to 10 cells per second, while the negotiated peak cell rate was set to 15 cells per second.

Looking at Figure 4-2 one sees that at the start of the simulation the average rate is higher than the negotiated average rate, and at about 1.2 seconds it reaches its peak value of 118% of the negotiated rate. This high value of the average cell rate is what contributes most to cells being dropped at that time. One thing to note is that the switch action SA is low when cells are dropped (in this case it drops right down to zero) and that the BUCKET fills up again because no cells are let through.

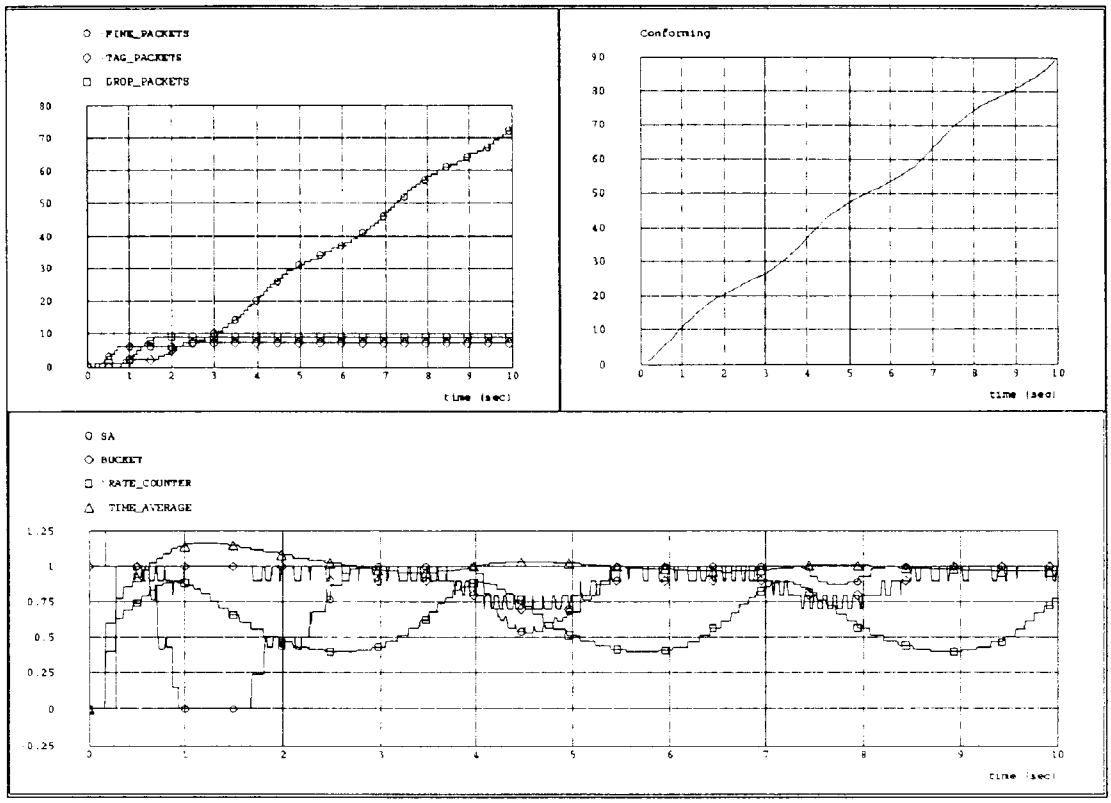



Figure 4-2 Case 2 - The source is mostly conforming.

As the simulation continues, the average rate settles down to the negotiated rate. Now that the average rate is within accepted limits, the current rate is less than the negotiated rate, and that the bucket is full, one would expect that the controller will let the cells through. This is shown in Figure 4-2 where the number of cells that were tagged and dropped does not change after two and a half seconds into the simulation.

The fact that some cells are considered to be non-conforming by the Fuzzy Logic Controller at the start of the simulation, is of some concern. The problem arises because the period after which the average rate is calculated is too short, and initially results in high average rates. A possible solution to this problem would be to choose a window period after which the average rate is calculated, but this would mean that possibly non-conforming cells would be let through at the start of the simulation. Since it is difficult to determine how long the connection will stay open it was decided not to implement the window period and stay with taking the time average instead.

Once cells are let through and the average rate is acceptable, one can see that the tokens in the Bucket decreases as the current rate increases at four and five seconds into the simulation, and increases again as the current rate decreases.

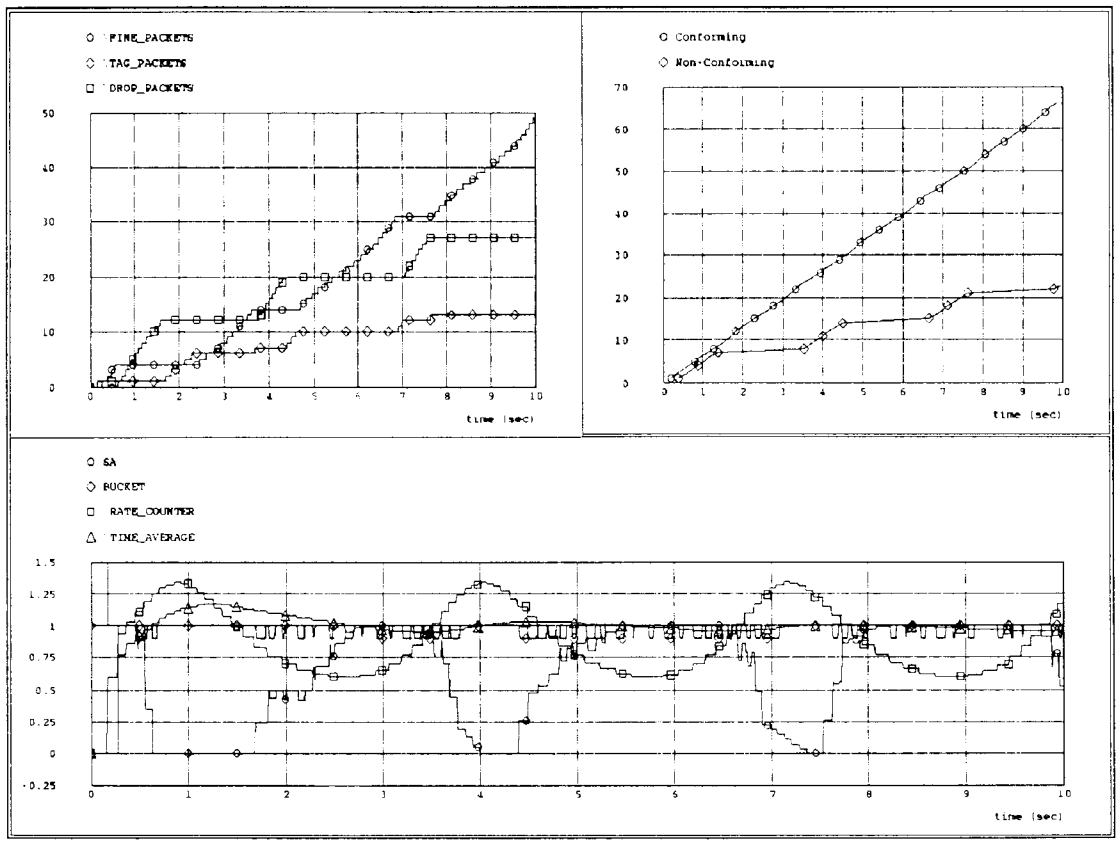



Figure 4-3 Case 2 - Mostly conforming source.

Figure 4-3 shows another case where the sinusoidally varying VBR source is used. The source is transmitting at the same rate as in Figure 4-2, but the negotiated peak cell rate has decreased from 15 to 10 cells per second. In this case the current cell rate goes up to a maximum of 135% and a minimum of 62% of the negotiated peak cell rate. Here one finds that during these periods when the average rate is too high, e.g. from 3.5 to 4.5 seconds, cells are tagged initially, dropped as SA reaches 0.5, and are tagged again when SA increases to 0.5. This shows the controller obeys the levels that were set to distinguish between FINE, TAG, and DROP cells, and thereby can distinguish between the three types of cells.

The overall impression for this case is that the Fuzzy Logic Controller is performing as expected, capable of dropping and tagging cells.

### 4.1.3 Case 3 - The source is non-conforming.

In this case the source was chosen to be well outside the negotiated range. A simple CBR source was thus considered sufficient for testing purposes. Two tests are presented here. In the first one the average rate is 10.1 cells per second while the negotiated rate is set to 10 cells per second, i.e. the average rate is 1% more than the negotiated rate. In the second test the average cell rate is set to 9.9 cells per second while negotiated rate is set to 2 cells per second, i.e. the average rate is set to approximately five times the negotiated rate. In both cases the peak cell rate was set to 10.1 cells per second, and the bucket size set to 10 cells.

The first test is shown in Figure 4-4. Here one sees that the cells are allowed through until the bucket has almost emptied, and that it takes 90 seconds[13] before cells are tagged. Thereafter cells are tagged at approximately 1.6 cells per second. The source is only allowed to transmit at a high rate until the controller detects that the bucket has dropped to a too low value. The controller is thus performing as expected.

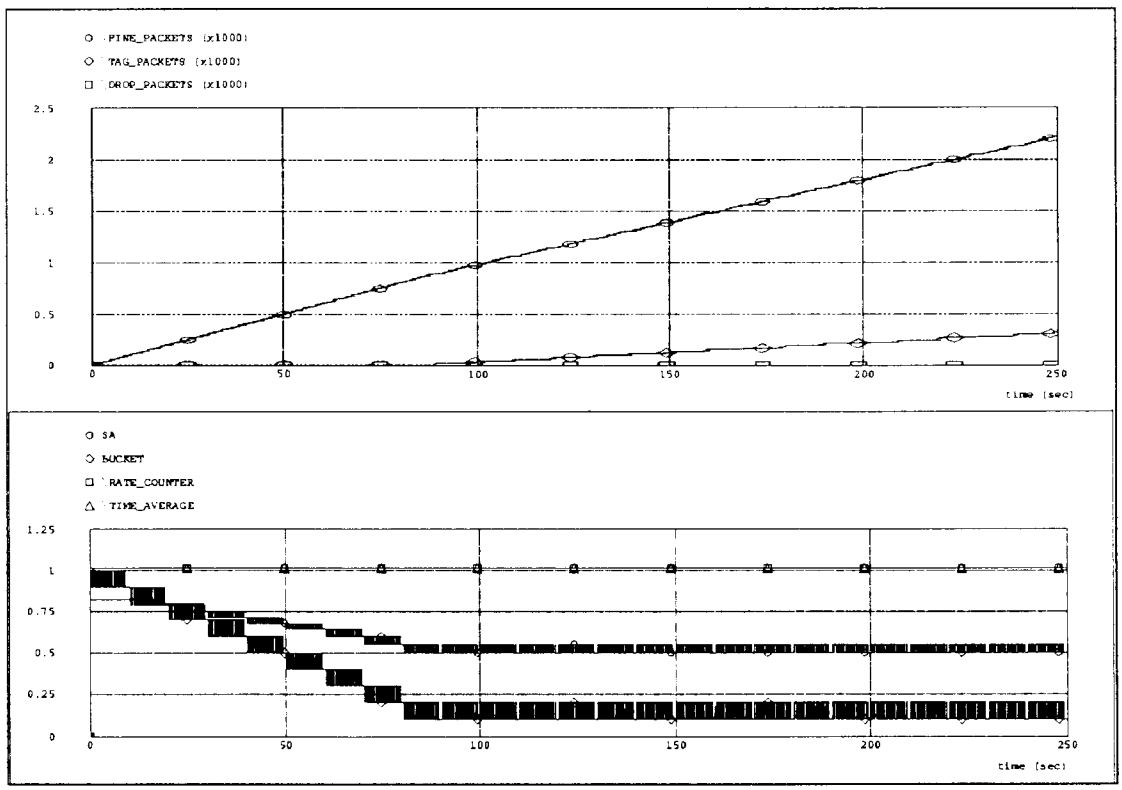



Figure 4-4 The source is non-conforming - test 1.

[13] (10 - 1) cells used / 0.1 cells per second overload = 90 seconds.

The results of the second test are shown in Figure 4-5. It shows that all cells are dropped, as expected, and that the switch action is down to zero. However, when comparing the results of the Fuzzy Logic Controller with that of the GCRA it is found that the GCRA outperforms the Fuzzy Logic Controller in the sense that it still allows cells through at the negotiated rate while discarding the rest. This is better than dropping all cells as is the case with the Fuzzy Logic Controller.

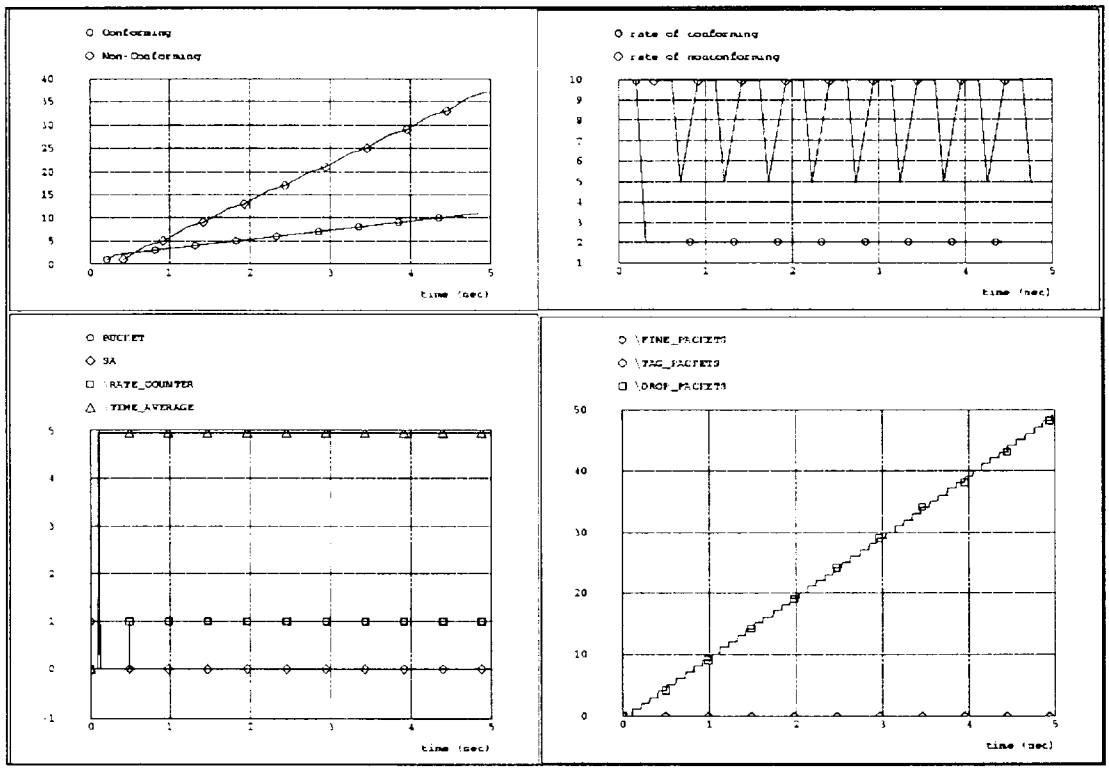



Figure 4-5 The source is non-conforming - test 2.

Even though the Fuzzy Logic Controller was performing as it was designed to, i.e. dropping cells that were totally non-conforming, no provision was made to allow a certain amount of those cells through during extreme cases of non-conformity.

The problem was that when the average rate became too high, cells were dropped even though the bucket was still full. Dropping cells meant that no tokens were taken from the bucket, and this caused the bucket to fill up again. But because the average rate stayed the same, the Fuzzy Logic Controller kept on producing the same output, namely to drop cells.

One can therefor see that the bucket has a much lower measure of importance when compared to that of the average rate. Ideally one would have expected cells to still be allowed through if there were any tokens left in the bucket. But this was not the situation in this case.

This now enables one to assess the performance of the controller. One finds that in this case the controller is performing as expected, but that it does not perform so well when compared to the GCRA.

### 4.1.4 The optimised fuzzy sets

After the optimisation period, it was found that the initial fuzzy sets had not changed by much and that the rule base had not changed at all. The initial estimates for the fuzzy sets and the rule base were thus not far off.

The only set that had changed by much was the SPOT-ON set of the Average Rate fuzzy variable. It was found that the narrower the base was made the faster the bucket settled to its final value (under steady state), and the faster the controller reached its final decision. It could however not be made too small because then the controller seemed to overshoot and oscillate around its control decision. A compromise was reached and the resultant new set can be seen in Figure 4-6.

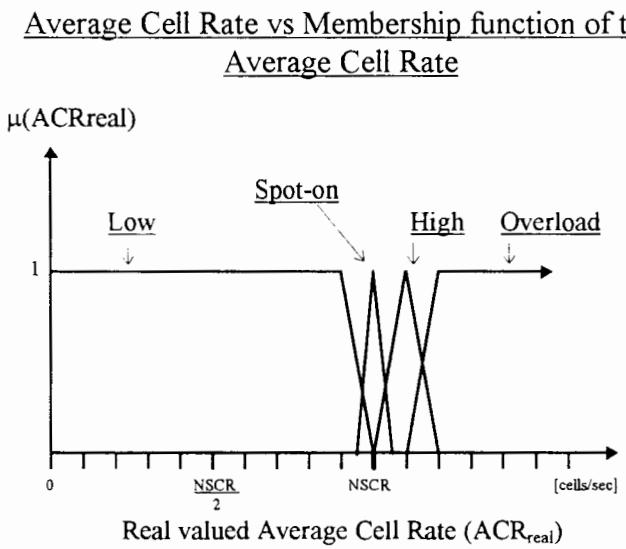



Figure 4-6 The optimised fuzzy set.

## 4.2 Evaluating the overall controller performance

The first two cases mentioned above show that cells are allowed through as long as the traffic from the source is not too excessive, and cells are tagged and dropped as the source increases its traffic.

The difficulty arises with the second test in the third case. Although the controller is performing as it was designed to, its control action my seem too excessive when viewed from the users point of view. If the source is transmitting at too high a rate it may not be fair to drop all of its traffic. If the source then had to choose between the Fuzzy Logic Controller and the GCRA, it would surely choose the GCRA because the GCRA maximises the amount of traffic that the source can send. If that was the case then the Fuzzy Logic Controller would be no better than the GCRA. Therefore, if the Fuzzy Logic Controller wished to compete with the GCRA, it would have to undergo some further design changes.

Two problems stand out when looking at the three test cases:

1. Cells are dropped even if there are tokens left in the bucket, and this causes too many cells to be discarded.
2. The average cell rate is calculated over too short a period of time at the start of the connection. This makes the average rate seem much higher than it does in the long run.

Solutions were sought to these two problems, and one was found that could possibly overcome both. The solution was to change the controller into a window based controller. This meant that a period would be set aside during which all cells that arrive are allowed through as long as there are tokens left in the bucket[14]. Cells that arrived when the bucket was empty would be dropped. The idea was to set the number of tokens in the bucket equal to the average expected number of cells in the chosen period, i.e. the number of tokens N in the bucket would be

$$N = Average\_rate * T \tag{14}$$

[14] A token is taken out of the bucket for each cell that passes through.

where T is the duration of the chosen window period. Therefore if too many cells arrived in the window period then the excess cells would be dropped. At the start of the next window period the bucket is filled up again and the cycle repeats itself.

This would take care of the average rate, but not of temporary high periods of transmission that will average out to the negotiated average rate over the long run. A method was needed to allow this type of bursts through.

Catania et al. presented a solution to this problem in [ 21 ]. They proposed to reward the source with more tokens as long as it stayed compliant to its traffic contract. This meant that if the source has been compliant for some time then it would have built up credit to support bursts that are bigger than the average rate. The amount of tokens are then slowly decreased if the source stays non-compliant for too long. This ensures that if the source is trying to send at a rate that is too high then eventually only the agreed upon number of tokens, N, are allowed through during each window period.

Changing the design into a window based controller also presented a solution to the second problem. Calculating the average rate after the window period meant that it would only be determined after several cells have entered the controller, and not as before when it was calculated after each incoming cell. This would reduce the high initial average rate that was found in the second test case discussed earlier. As time passes, however, the average rate calculated using the window period will equal the average rate calculated after the arrival of each new cell. This happens because both methods use the total number of cells that arrived up to the moment of evaluation to determine the average rate. In the long run the average rate for the window based controller would thus be the same, and could still be used as before.

Using a window based controller would however also cause the current rate to be averaged over the window period. This meant that a possible short burst at a high rate might be missed by the controller if the rest of the window period had a low incoming rate. But, considering the new role of the token bucket, this would no longer present a problem. Only the number of cells for which there are tokens in the bucket would be allowed through.

At that stage it was not known exactly how the "averaged" current rate and the window period T would affect the performance of the controller. It was decided to postpone

this evaluation until some simulation results, using the design changes, could be obtained.

Introducing these changes also implied that the fuzzy sets and the rule base of the controller would have to be changed, and would effectively produce a new fuzzy logic controller. One is then faced with a choice:

1. either stay with the current controller and allow cells to be dropped if the source is completely non-conforming, or
2. continue with the new design and see whether it can outperform the GCRA.

The author decided to take the next step and modify the current design to see how well it would perform. This new, and final design, is presented in the next chapter.

# 5. THE MODIFIED FUZZY LOGIC CONTROLLER

The previous chapter showed that when the traffic source was conforming or marginally non-conforming, the controller was able to decide between tagging and dropping non-conforming cells, and letting conforming cells through unmodified. One also found that the Fuzzy Logic Congestion Controller did not perform as well as the GCRA in cases where the traffic arrived at an average rate much higher than the negotiated average rate. In such cases of extreme traffic contract violation, the controller classified all of the traffic as non-conforming and dropped all of the cells as a result.

Although it is the prerogative of the network, and in this case the controller, to decide what to do with non-conforming traffic, it was felt that the source would receive a better quality of service if at least some of its traffic was allowed to go through. This meant that the controller presented in the previous two chapters had to be modified in order to provide a better quality of service to the source. It was also hoped that the new modifications would enable the controller to outperform the GCRA. Simulations of the new controller would show whether or not this would be the case.

This chapter presents the modifications that were suggested to improve the controller. The ideas behind the functioning of the controller did not change by much and therefore the modifications that will be discussed here, will not be handled to the same level of detail as before. This means that the functioning of the new or modified modules will be explained in detail, but concepts like what a rule base is used for will not be repeated.

The layout of this chapter is similar to that of chapter three, and was done to enable the reader to quickly see how the new design differs from the old. The traffic parameters and the control action are discussed again as this is central to the design of the controller. This is followed by descriptions of the modified design modules, and how the fuzzy sets and rule base changed in order to convert the controller into a window based controller.

## 5.1 The Traffic Parameters

Three traffic parameters were used in the previous controller, namely the Peak Cell Rate, Average Cell Rate, and Burst Threshold. The first two are used again in the new controller, while the third, the Burst Threshold, is replaced by a parameter called the Initial Bucket Size (IBS).

As mentioned earlier, the new controller is a window based controller and it allows a certain amount of cells through during each window period. The number of cells that are allowed through depends on how the source has performed in the past, and is equal to the amount of credit that the source has earned. The controller awards the source with more tokens if it keeps its average rate lower than or equal to its negotiated average rate, and penalises the source by decreasing its credit when it goes above the negotiated rate. The idea is to give the source just enough credit when it is non-conforming, so that only the negotiated average number of cells per window is allowed through. More credit is added when the rate increases again.

Now that one knows that the amount of credit the source owns can vary, the question arises of what value should it be assigned at the start of the connection. If the credit assigned to the source is too low, then an initial burst of traffic that is higher than the average rate will be dropped when it would have been conforming in the longer run. On the other hand, if the initial credit is too high then too many cells will be allowed through if the connection will terminate shortly after the end of the burst.

One option is to set the Initial Burst Size equal to the Burst Threshold as in the case of the GCRA. This method could however mean that too many cells will be allowed through, since the Burst Threshold can be large if the negotiated PCR is large[15].

Another option is to set the Initial Burst Size equal to a fixed multiple of the average number of cells that are expected during a window period. If this multiple is initially too low then too many cells may again be dropped. This method also presents the controller with less information about the source characteristics than if the source supplied this value as a traffic parameter. Also, a fixed multiple may not always work for

[15] Burst Threshold = Negotiated PCR * Burst Duration

all traffic sources when considering the range of parameter values that can be assigned to VBR sources.

The author decided to keep the IBS as a source traffic parameter so that the source could have a say about how its initial burst of traffic should be handled. The GCRA also determines its buffer size by requesting the burst size from the source when policing the average rate. A problem with the GCRA, however, is that if the burst size specified is too large then it means that the GCRA (which is related to the leaky bucket algorithm) would be over-dimensioned and "decrease the ability of the mechanism to detect real long-term parameter violations" [ 2 ]. The new Fuzzy Logic Controller will not have this problem over the long run since the credit the source owns will decrease if the traffic from the source stays non-conforming.

Allowing the source to specify an unlimited Initial Burst Size, is also not good. If the source started off by being non-conforming, then it would take the controller some time to decrease the credit to a level where only the agreed upon average number of cells per window would be allowed through. Thus it is essential to limit the maximum Initial Burst Size to keep the reaction time of the controller low.

It is also possible that the credit the source has earned can grow indefinitely if it keeps its average rate low. If after a long period the source then suddenly starts to become non-conforming, it will take the controller some time to decrease the credit to an acceptable level. Thus it is also important to limit the maximum amount of credit that the source can own.

During the numerous tests that were performed on the new controller, it was found that a conforming source seldom reached a credit level that was higher than nine times the average expected number of cells per window. A maximum of nine times the expected average number of cells per window period was chosen as an upper limit. An option was also built into the controller where this upper limit could easily be changed to a lower value if tighter control was required over the source.

The traffic parameters used are thus:

1. the peak cell rate,
2. the sustainable cell rate and
3. the initial burst size.

The network is also allowed the option of determining what level of control should be enforced on the source. This can be done by instructing the controller to increase or decrease the upper credit limit of the source.

## 5.2 The Control Action

The objective of the new controller is to reward the source with more credit when it is conforming, and to penalise the source by decreasing its amount of credit when it is non-conforming. Thus the control action is to either increase or decrease the credit level, or to leave the credit level unchanged.

One can no longer think of the controller as tagging, dropping, or leaving cells unmodified. The modified Fuzzy Logic Controller was designed to see how well it would compare with the GCRA, and since the GCRA only classifies cells as either conforming or non-conforming, it was decided to drop the option of tagging cells from the control decision of the Fuzzy Logic Controller. In this manner direct comparisons can be made between the two congestion control methods.

It is now the duty of the controller to decide by how much the credit level must change at the end of each window period. Large positive increments are taken if the average rate is very low, while large negative increments are taken if the average rate is much higher than the negotiated rate. The controller however makes sure that the maximum credit level is not exceeded if positive increments are used, and is designed to bring the level back up to the expected level[16] if negative increments take it below that level. This ensures that only the accepted number of cells is allowed through when the source is non-conforming.

Once the controller determines the size of the increment, it is then added to the amount of credit that was assigned at the start of the previous window period. The updated credit level is then used to determine how many cells will be allowed through during the current window period.

[16] The expected level is equal to the number of cells arriving during the window period at the negotiated average rate.

## 5.3 The Modified Fuzzy Logic Congestion Controller

The new controller consists of the Rate Counter, the Switch Module and the Fuzzy Logic Controller. Only three modules are needed in this case because it was decided to simplify the design even further. Several design changes were made:

1. The Input and Output modules were removed from the design and placed directly into the Switch module.
2. The Leaky Bucket variable (or Up-Down counter) was placed inside the Switch Module as well, because it now works in much closer co-operation with it. This variable now acts as the counter that determines how many cells are still allowed to go through to the network during the window period, and thus keeps track of how much credit is still remaining to the source. This counter is decremented by one unit for each cell that is allowed through.
3. The Switch module only allows cells through while the source has credit left, and drops all cells once the credit has run out. It is now also the duty of the Switch module to inform the Rate Counter when a new cell arrives, since the Input module has been incorporated into its design and can no longer perform this function.
4. The Fuzzy Logic Controller contains an additional variable that is used to store the last credit level that was sent to the credit counter inside the Switch module. It is called the Bucket variable. At start-up this value is set to the Initial Burst Size, and is also forwarded to the Switch module to set the initial credit level. At the end of each window period this value is updated (as explained earlier) and forwarded again to determine how many cells will be allowed through to the network during the current window period.

Figure 5-1 shows the layout of the new controller. From the changes mentioned above one can see that outwardly the design has not changed much. It is internally, in the fuzzy sets and rule base, that the greatest change has taken place. Two additional modules are also shown that are not considered part of the controller. The first is the traffic source that is used to test the controller with, and the second is the GCRA module against which the performance of the new Fuzzy Logic Controller is be compared. The source in this case is a VBR source that simulates an MPEG source.

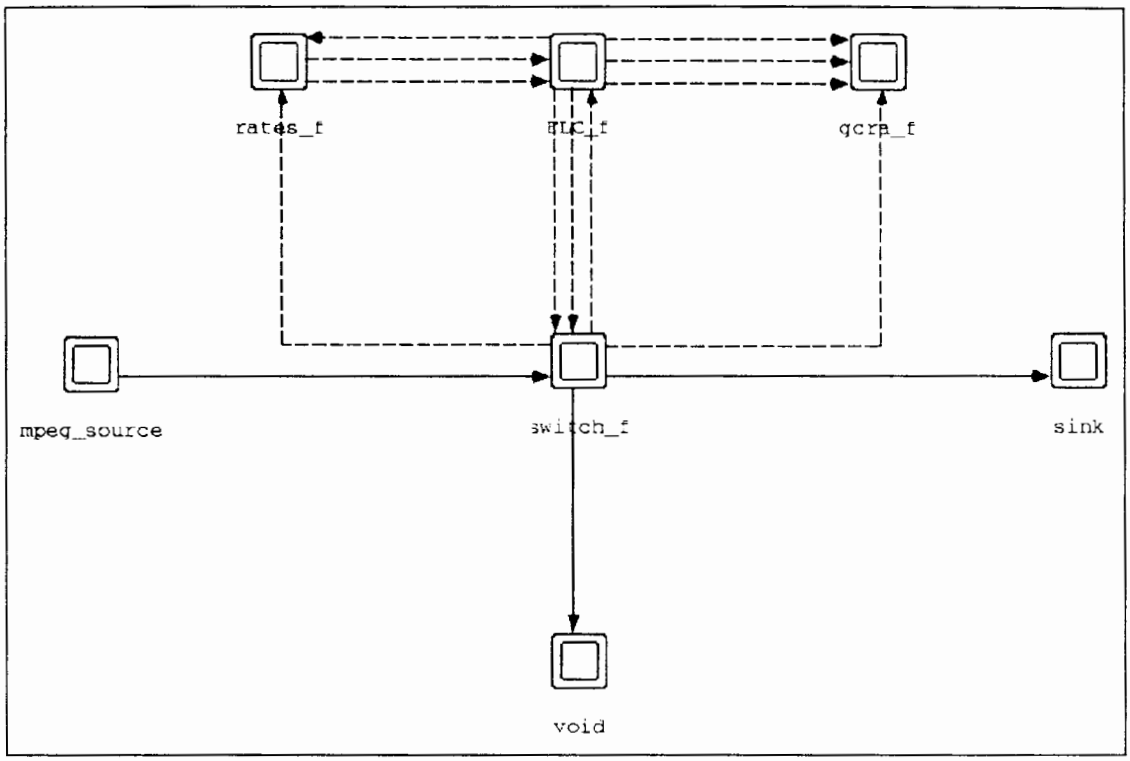



Figure 5-1 The Modified Fuzzy Logic Congestion Controller.

This traffic source will be explained in more detail in the next chapter where the simulation of this controller will be discussed.

Cells arrive from the source and are passed to the Switch module. Every time a cell arrives the Switch module notifies the rate counter and the GCRA module of its arrival, and passes the cell through to the sink module[17] if the source has any credit left. If all the credit has been used then the cell is dropped by routing it to the void module.

This concludes the layout of the modified Fuzzy Logic Controller. The only design changes that still have to be discussed are the new fuzzy sets and rule base that determine how the controller will behave. These are discussed in the next two sections.

[17] The sink module represents the rest of the network.

## 5.4 The Fuzzyfication and Defuzzyfication Process

The new fuzzy variables consist of the Current Cell Rate, Average Cell Rate, and the Bucket variable that stores the amount of credit that is assigned at the start of each window period.

### 5.4.1 The Average Rate and Current Cell Rate

The Current Cell Rate is defined as the number of cells that arrived during the window period, divided by the duration of the window period, while the Average Cell Rate is defined as the total number of cells that arrived from the start of the connection to the end of the current window period, divided by the current duration of the simulation. It was decided to assign the same fuzzy sets to both variables in order to later simplify the optimisation of those sets. The sets were also normalised to the Negotiated Average Rate so that the same sets could be used for different traffic sources. The layout of these two fuzzy sets is shown in Figure 5-2.

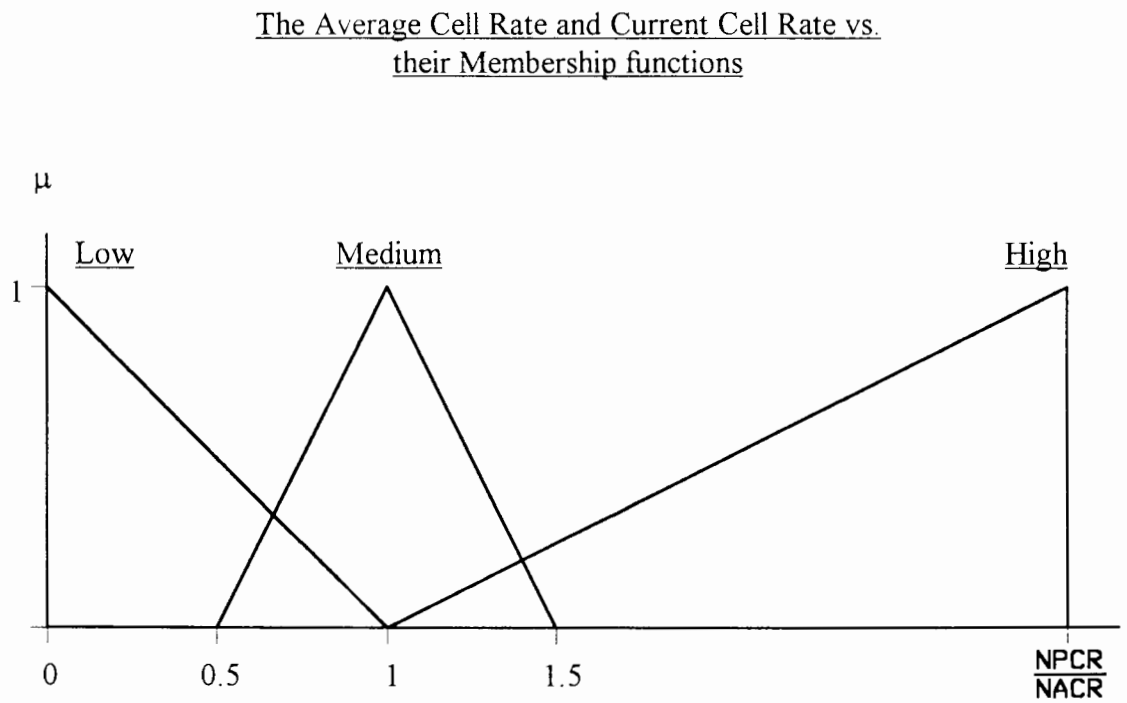



Figure 5-2 The modified fuzzy sets for the Current and Average Cell Rates, normalised to the Negotiated Average Cell Rate (NPCR/NACR >= 1.5).

Looking at Figure 5-2, one can see that the MEDIUM fuzzy set may extend past the HIGH fuzzy set if the negotiated PCR is less than 1.5 times the negotiated ACR. Therefor, to prevent that this overlap ever occurs, the fuzzy sets shown in Figure 5-3 are used to take care of this situation if it ever arises.

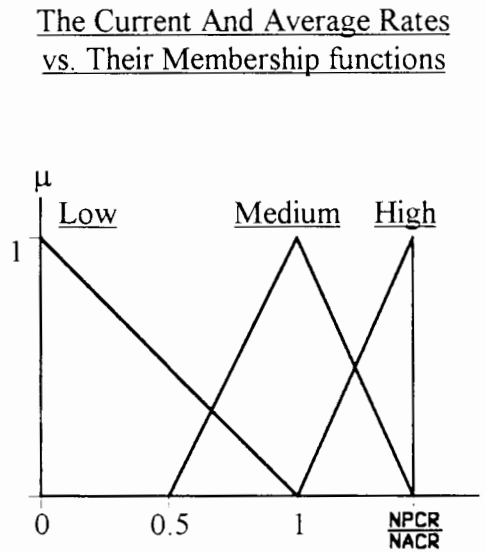



Figure 5-3 The modified fuzzy sets for the Current and Average Cell Rates, normalised to the Negotiated Average Cell Rate (NPCR/NACR < 1.5).

### 5.4.2 The Bucket variable

Figure 5-4 defines the Bucket variable. One sees that the MEDIUM set is centred on $N_e$, the expected average number of cells per window period at the negotiated average cell rate. $N_t$ represents the number of cells that are expected if the source sends at its negotiated peak rate for the duration of the window period. $N_{max}$ is the upper limit on the amount of credit that the source can own.

Figure 5-5 is used in the event that $N_t$ is ever less than 1.5 times $N_e$.

The Bucket variable vs. its Membership function.

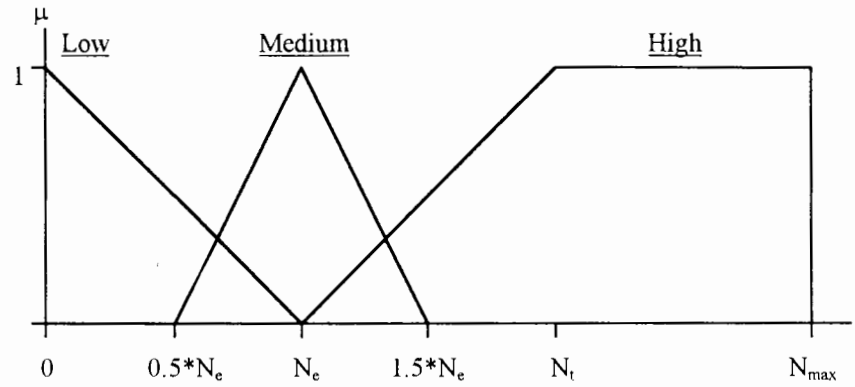



Figure 5-4 The fuzzy sets defining the Bucket variable (Nt >= 1.5*Ne).

The Bucket variable vs. its Membership function.

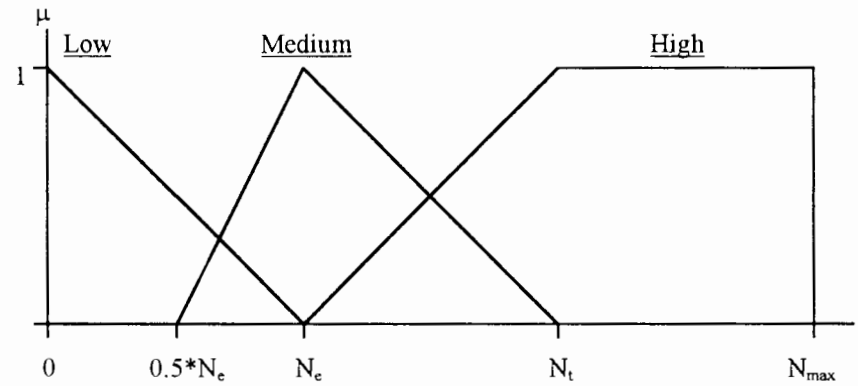



Figure 5-5 The fuzzy sets defining the Bucket variable (Nt < 1.5*Ne).

### 5.4.3 The Switch Action

The Switch action determines by how much the credit will be incremented (or decremented) in the next window period. This increase (or decrease) should however be small enough to provide a smooth control action from one window period to the next, but simultaneously large enough to present a fast response time to policing a non-conforming source. Keeping this in mind, it was felt that the maximum change should be less than the expected average number of cells per period ($N_e$). A compromise was reached by setting the maximum possible credit change per window to a third of $N_e$. This then defines the extreme limits of the fuzzy sets used for the switch action.

The variable was divided into five sets, two to make negative changes, two to make positive changes, and one to keep the credit level unchanged. These sets are called NEGATIVE BIG, NEGATIVE SMALL, ZERO, POSITIVE SMALL and POSITIVE BIG, and are defined in Figure 5-6. In chapter three it was suggested that the number of fuzzy sets per variable be kept as low as possible, and an upper limit of three sets per variable was selected. This was done to keep the size of the rule base as small as possible. The size of the rule base is, however, not affected by the number of sets in the output variable, and therefor a total of five sets for the output variable remains acceptable.

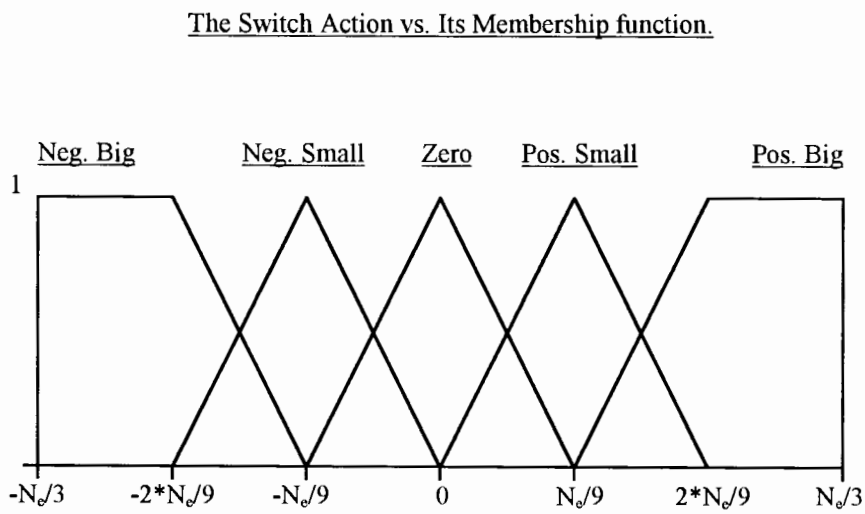



Figure 5-6 The fuzzy sets defining the modified Switch Action.

## 5.5 The Rule Base

The objective of the new rule base is to reward the source with more credits if it is conforming, and reduce the credit if the source is non-conforming. The rule base is divided into three tables: one for when the ACR is LOW, one for when it is MEDIUM, and one for when it is HIGH.

| BUCKET / CCR | LOW | MEDIUM | HIGH |
|---|---|---|---|
| LOW | PB | PB | PB |
| SPOT-ON | PB | PB | PS |
| HIGH | PB | PS | Z |

Table 3 The Rule Base - ACR is Low.

| BUCKET / CCR | LOW | MEDIUM | HIGH |
|---|---|---|---|
| LOW | PB | PB | PS |
| SPOT-ON | PS | PS | Z |
| HIGH | Z | Z | NS |

Table 4 The Rule Base - ACR is Medium.

| BUCKET / CCR | LOW | MEDIUM | HIGH |
|---|---|---|---|
| LOW | PB | PB | NS |
| SPOT-ON | PS | PS | NB |
| HIGH | PS | Z | NB |

Table 5 The Rule Base - ACR is High.

The Rule Base is completely full, and this means that all the possible states of system are covered. One can however reduce the number of rules if one knows what the Initial

Burst Size will be. If the IBS is bigger than the expected average number of cells per window period at the start of the connection, then the first column in Table 3 can be removed, because these rules will never be activated.

This concludes the modifications to the controller. Its performance is evaluated in the next chapter.

# 6. SIMULATION OF THE MODIFIED FUZZY LOGIC CONTROLLER

The ATM-Forum has accepted the GCRA as a possible method for how Usage Parameter Control can be performed in ATM networks. It is easy to implement, and can be used to police both the average and peak cell rate of a connection. Although it is one of the most widely used algorithms, the network provider is not obliged to use it [ 18 ]. But, because of its performance and simplicity, it is often used as a "yardstick" against which other UPC algorithms are measured. In this chapter the GCRA is used to determine how well the modified Fuzzy Logic Congestion Controller performs when policing a VBR source.

One of the biggest problems one experience when designing a preventive congestion controller in a simulation environment, is that it is difficult to come up with a traffic source with which to test the controller. A VBR source was sought that could be used to extensively test the performance of both the GCRA and the modified Fuzzy Logic Congestion Controller. The author felt that the sinusoidally varying VBR source that was used to test the previous design, was too regular and periodic to be used for this test. Therefore, an alternate source with a greater degree of variation was sought.

Several tests were performed on a source that had a gaussian distributed cell rate. Even though various mean and variance values were tested, it was found that the characteristics of the source were not acceptable. If the variance of the source was too low, then the average rate did not vary by much, and if the variance was too high then the current cell rate was too random. There was also some doubt on how realistic it was to use a VBR source with a gaussian distributed cell rate.

It was then suggested that the author try to implement a VBR source with the characteristics of an MPEG-encoded video stream. An MPEG data stream would not only provide the desired traffic characteristics, but also represent a traffic source that had practical application in the real world. The performance of the two control methods could then be compared to see how well they perform when faced with a real-world traffic problem.

This chapter presents how the MPEG source was generated, and how the GCRA and the modified Fuzzy Logic Controller were configured to police it. Simulation results of both control methods are presented to show their performance.. The chapter then ends off with an evaluation of how well the two control methods policed the source, and particularly how well they performed against each other.

## 6.1 The MPEG-encoded video source

Video over ATM is one of the classes of service that will be available on ATM networks. In fact, hardware[18] already exists that enables one to do video-conferencing over an ATM network. When considering the high transmission rates needed to transmit video one finds that ATM is an ideal carrier for such data.

However, methods are constantly sought that will enable one to increase the throughput of today's communication systems. If more information can be sent over the same communication medium, it means that the medium will be better utilised, and from an economics point of view be more cost effective for the service provider. For example, if a company providing video on demand over an ATM network found that it could present 50% more video channels on the same network if some form of compression method was used, then it would have an edge over its competitors. Better network utilization means that more clients can have access to the resource than before, and this could mean a greater profit at the end of the day.

MPEG is a compression algorithm that has been developed by the Motion Picture Experts Group, that has recently gained considerable attention for its ability to encode streams of video data. With MPEG one finds that one is no longer restricted to sending all the information in every frame of a video source. When looking at a video sequence one finds that there are often scenes where there is not much difference between the

[18] Nemesys produces the AVA300 and ATV300 units which allow the real-time encoding and decoding of video streams for video-conferencing over an ATM network. The author has participated in testing this equipment on Telkom South Africa's ATM Pilot network. Audio and Video from a hand held video camera was encoded by an AVA300 unit at the University of Cape Town and transmitted to the University of Pretoria over the pilot network where it was decoded using an ATV300 unit that displayed the video image on a normal television screen. Similar sources were sent from the University of Pretoria and decoded at the University of Cape Town. The resultant audio and video were of such high quality that it equalled that of normal television transmissions, and by using this technology video-conferencing sessions was possible. The system has a wide range of applications, one of which is distance learning. This is another application that has been tested between the two universities.

images that are presented. Take for example a video sequence showing the clouds in the sky. Very little new information is presented after one sees the first few frames of the cloud sequence, and it is the this redundancy of information that the MPEG compression scheme takes advantage of.

What MPEG does is to take a group of pictures (GOP) (or frames) and encode them so that the amount of redundant information presented in the later frames in the GOP are as low as possible. This is done by encoding the frames as Intra (I), Predictive (P), or Bi-directional frames (B). A series of I , P and B frames is then used to form the compression pattern of the GOP. Only one I frame is used per compression pattern, and is placed at the start of the pattern. One finds that I-frames generally contain the most data in the compression pattern because this is when most of the information contained in the GOP is presented, while the P and B frames contain much less.

The size of each compressed video frame can thus vary when using MPEG compression, and depends on how actively the scenes change during the video sequence and the type of compression that was chosen. It is not unusual to find that frames can vary from 300 Kbits, to as low as only a few Kbits during a single scene[19].

The GOP consists of a number of frames in which only one I frame exists in a group of N frames, and the distance D between an I frame and the next P frame also equals the distance between consecutive P frames. If one then inserts B frames where I and P frames do not occur, then the following sequence, IBBPBBPBBPBBPBB, is obtained for N=15 and D=3, while IBBPBB is obtained for N=6 and D=3. Looking at these sequences one finds that there are a lot more P and B frames than I frames in a GOP. And when one considers that the bulk of the data generated as a result of the compression pattern can be found during the I frames, one finds that a MPEG video stream can have an extremely high ratio of peak cell rate versus average cell rate. It is this type of sources which are particularly difficult to police.

Please refer to [ 20 ] for more details on the MPEG standard.

The traffic model by Krunz and Hughes was chosen to implement the MPEG-coded VBR source [ 19 ]. They present "a model for an MPEG traffic source ... in which

[19] A scene is defined as a video sequence in which no sudden view changes take place.

frames are generated according to the compression pattern of the captured video stream. For each frame type, the number of cells per frame is fitted by a lognormal distribution whose parameters are determined by the frame type." Their compression pattern is 15 frames long (N=15, M=3) where they chose `IBBPBBPBBPBBPBB` as their compression pattern. Each I, P or B frame is transmitted in one thirtieth of a second where the cells are equally spaced during the transmission of each frame. This gives a transmission rate of 30 frames per second.

Krunz and Hughes took 23 minutes of video from the movie *The Wizard of Oz* and developed their model of an MPEG traffic source. They then simulated an ATM multiplexer with traffic sources derived from the measured video source as well as from their traffic model. Their results show that the "queuing performance in both cases [were] found to be relatively close."

Their results were thought sufficient to support the choice of including their MPEG traffic source as the test source for the modified Fuzzy Logic Congestion Controller.

### 6.1.1 Simulating the MPEG Source

Figure 6-1 shows a 100 second simulation of the MPEG traffic source in OPNET®. It shows the frame size, the current and average cell rate, and the probability density of the frame size and the current cell rate. The current and average rates were calculated using a window period of 0.5 seconds.

The graph of the frame size in Figure 6-1 compares well with what Krunz and Hughes presented in [ 19 ], and when comparing the shape of the probability density function of the frame size with that displayed in [ 19 ], one finds that they are almost identical. This is an indication that the source is performing as predicted.

The next figure, Figure 6-2, is an enlargement of the graph of the frame size shown in Figure 6-1, taken from zero to almost thirty seconds into the simulation. Here one sees that there are smaller peaks between the more regular bigger peaks. The smaller peaks are the P and B frames of the source, while the bigger peaks are the I frames. It is unfortunately not possible to count the number of frames between each I frame in the figure, due to the low resolution of the picture. The author was however able to enlarge the graph on the SUN workstation (on which OPNET® executed) to verify that there were 14 smaller peaks (i.e. P and B frames) between each I frame. It was however not possible to reproduce that enlargement here.

From this test one can conclude that the OPNET® MPEG traffic source compares well with the model presented by Krunz and Hughes, and that it is acceptable to use this traffic source as the MPEG-encoded VBR source for testing the GCRA and modified Fuzzy Logic Congestion Controller.

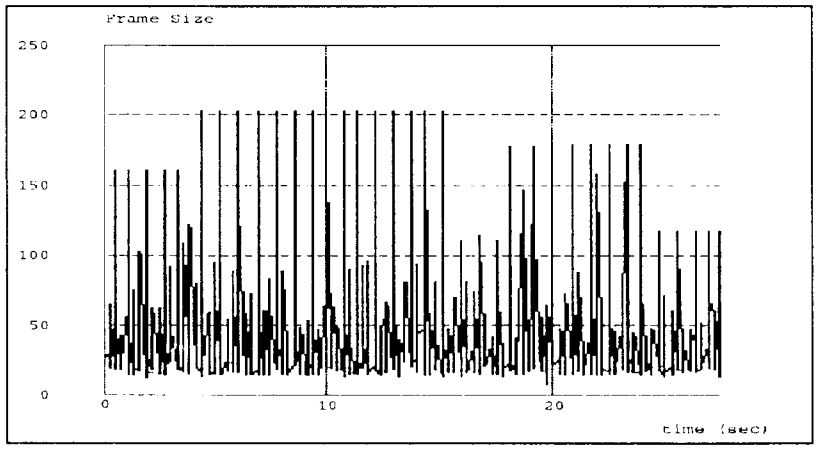



Figure 6-2 An enlarge view of the frame size of the MPEG source.

### 6.1.2 Obtaining the traffic parameters of the source.

The next objective is to extract the traffic parameters from the source so that the controllers can know what parameters to police. This is done by running each simulation twice. OPNET® allows the user to set the seed value for the random numbers that it will generate during the simulation. One is thus assured that the sequence of random numbers generated will be exactly as in a previous simulation, providing that the seed value is kept constant between simulations.

The first run is used to collect the relevant data about the source, namely the peak cell rate, the average rate, and the burst size. While this takes place all other data is ignored. The first run is thus used to characterise the source.

The peak cell rate is easy to obtain. Simply keep track of the maximum frame size during the first run and divide it by thirty at the end of the simulation. It is valid to divide the maximum frame size by thirty because each frame is sent in one thirtieth of a second, and the cells of a frame are equally spaced during the transmission of that frame.

The average cell rate is determined by counting the number of cells that arrive at the Switch module, divided by the duration of the simulation.

The peak burst duration was chosen as the time it takes to transmit one frame. Therefor the burst size equals the product of the burst duration and the peak cell rate.

The data collected during the first run is then used to initialise the policing algorithms at the start of the second run. In this way one can find the exact values to specify for the average and peak cell rate of the source during a particular simulation.

## 6.2 Configuring the policing algorithms

The GCRA and the modified Fuzzy Logic Congestion Controller are both initialised at the start of each simulation.

### 6.2.1 The GCRA

Initialising the GCRA is straightforward. The negotiated PCR, ACR and peak Burst Size is set to the peak cell rate, average cell rate and burst size respectively that were obtained during the first run. Since one is not considering the cell delay variation here, that value is set to zero. The Burst Size for the GCRA is calculated using

$$BS = \frac{NPCR}{frame\_rate} \tag{15}$$

### 6.2.2 The modified Fuzzy Logic Congestion Controller

More consideration is needed when initialising the fuzzy controller.

The peak and average cell rates that were obtained during the first run are used to set the negotiated peak and average rate in the controller.

The Initial Burst Size was set to 1.5 times the expected average number of cells during the window period. This low value was chosen to show what the effect would be if the initial burst is higher than the amount of credit assigned at the start of the connection.

The upper limit of the amount of credit that the source can own was set to nine times the expected average number of cells per window for reasons presented in the previous chapter.

The size of the window period T was determined by looking at the characteristics of the source. Catania et al. presented a solution to this problem in [ 21 ]. They suggested that the window period be set to the time it takes to transmit the number of frames in

the compression pattern, and that the window period be synchronised to the start of the I frame. Considering the credit is updated at the start of each window period, this means that when the I frame arrives, the maximum amount of credit assigned to the source will be available for its use. In this way the most important frames, the I frames, have the greatest probability of being allowed through without one of their cells being dropped. For the source this means that the most important frame of the compression pattern will have the highest probability of being allowed through. Due to the characteristics of MPEG-encoding, if the I frame (or cells in the I frame) is lost, then the compression sequence can not be regenerated. The decoding mechanism then has to wait for the arrival of the next I frame, while the P and B frames are effectively discarded. This looks on the decoded stream (i.e. the video monitor) as if the picture has frozen and does not restart until the next I frame is received. The frozen-picture effect is not as severe when a P or B frame is the first frame in the sequence that is discarded.

The window period was thus set to how long it takes to transmit one compression sequence, i.e. the number of frames per compression pattern divided by the frame rate. Since the frame rate is 30 frames per second and there are 15 frames per compression pattern, T equals 0.5 seconds.

Now that both control methods are initialised one can proceed with the simulations.

## 6.3 The Simulation Results

Three test cases were simulated in order to see how well the GCRA and modified fuzzy controller performed with the MPEG traffic source. The test cases were chosen to see if the controller would be able to:

1. handle a conforming source, and
2. see how close it comes to the GCRA when the source is no longer conforming.

Ideally no cells would be dropped in the case of a conforming source, and only the agreed upon rate would be allowed through in the non-conforming case.

Two levels of non-conformity were tested. The first had an average cell rate 1.2 times bigger than the negotiated rate, while the second increased the average cell rate to 1.5 times the negotiated average rate. It was easy to specify exactly what the negotiated ACR should be in order to set these limits of overload. Once the characteristics of the source were obtained after the first run, the negotiated ACR was calculated by dividing the average cell rate by the overload factor m, i.e.

$$NACR * m = ACR \tag{16}$$

and thus,

$$NACR = \frac{ACR}{m} \tag{17}$$

This value of the negotiated ACR was then used to initialise the two control methods.

For example, during one simulation it was found that the source had an average rate of 800.1 cells per second and a peak rate of 7590 cells per second. The negotiated ACR was thus set to 800.1, 666.8 and 533.4 cells per second when m was equal to 1, 1.2 and 1.5 respectively, while the negotiated PCR stayed the same for all three cases. The Initial Burst Size was set to 600, 500, and 400 cells per second for m equal to 1, 1.2 and 1.5 respectively, while the Burst Size of the GCRA was set to 248 cells.

A total of thirty simulations were run to test the performance of the controllers, ten for each overload factor. Table 6 presents the variables that were used for those simulations. Each test is numbered and has a corresponding graph number in Appendix C which shows the simulation results for that particular test.

Only three graphs are shown in the following three sections due to the volume of data generated during the simulation runs. All the simulation results can however be found in Appendix C.

| Test | m | Seed | NPCR | Number of cells | ACR | NACR | IBS (fuzzy) | BS (GCRA) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 100 | 7080 | 78113 | 781.1 | 781.1 | 585.8 | 236 |
| 2 | 1.2 | 100 | 7080 | 78113 | 781.1 | 650.9 | 488 | 236 |
| 3 | 1.5 | 100 | 7080 | 78113 | 781.1 | 520.7 | 390.5 | 236 |
| 4 | 1 | 200 | 7860 | 80239 | 802.4 | 802.4 | 601.8 | 262 |
| 5 | 1.2 | 200 | 7860 | 80239 | 802.4 | 668.7 | 501.5 | 262 |
| 6 | 1.5 | 200 | 7860 | 80239 | 802.4 | 534.9 | 401.2 | 262 |
| 7 | 1 | 300 | 8250 | 75363 | 753.6 | 753.6 | 565.2 | 275 |
| 8 | 1.2 | 300 | 8250 | 75363 | 753.6 | 628 | 471 | 275 |
| 9 | 1.5 | 300 | 8250 | 75363 | 753.6 | 502 | 376 | 275 |
| 10 | 1 | 400 | 6840 | 75014 | 750.1 | 750.1 | 562.6 | 228 |
| 11 | 1.2 | 400 | 6840 | 75014 | 750.1 | 625.1 | 468.8 | 228 |
| 12 | 1.5 | 400 | 6840 | 75014 | 750.1 | 500.1 | 375.1 | 228 |
| 13 | 1 | 500 | 6600 | 76986 | 769.9 | 769.9 | 577.4 | 220 |
| 14 | 1.2 | 500 | 6600 | 76986 | 769.9 | 641.6 | 481.2 | 220 |
| 15 | 1.5 | 500 | 6600 | 76986 | 769.9 | 513.3 | 385 | 220 |
| 16 | 1 | 600 | 7590 | 80174 | 801.7 | 801.7 | 601.3 | 253 |
| 17 | 1.2 | 600 | 7590 | 80174 | 801.7 | 668.1 | 501.1 | 253 |
| 18 | 1.5 | 600 | 7590 | 80174 | 801.7 | 534.5 | 400.9 | 253 |
| 19 | 1 | 700 | 7440 | 80012 | 800.1 | 800.1 | 600.1 | 248 |
| 20 | 1.2 | 700 | 7440 | 80012 | 800.1 | 666.8 | 500.1 | 248 |
| 21 | 1.5 | 700 | 7440 | 80012 | 800.1 | 533.4 | 400.1 | 248 |
| 22 | 1 | 800 | 7500 | 79137 | 791.4 | 791.4 | 593.6 | 250 |
| 23 | 1.2 | 800 | 7500 | 79137 | 791.4 | 659.5 | 494.6 | 250 |
| 24 | 1.5 | 800 | 7500 | 79137 | 791.4 | 527.6 | 395.7 | 250 |
| 25 | 1 | 900 | 7710 | 77912 | 779.1 | 779.1 | 584.3 | 257 |
| 26 | 1.2 | 900 | 7710 | 77912 | 779.1 | 649.3 | 487.0 | 257 |
| 27 | 1.5 | 900 | 7710 | 77912 | 779.1 | 519.4 | 389.6 | 257 |
| 28 | 1 | 1000 | 8760 | 82237 | 822.4 | 822.4 | 616.8 | 292 |
| 29 | 1.2 | 1000 | 8760 | 82237 | 822.4 | 685.3 | 514 | 292 |
| 30 | 1.5 | 1000 | 8760 | 82237 | 822.4 | 548.3 | 411.2 | 292 |

Table 6 The Simulation Parameters.

### 6.3.1 Case 1 - Conforming Source

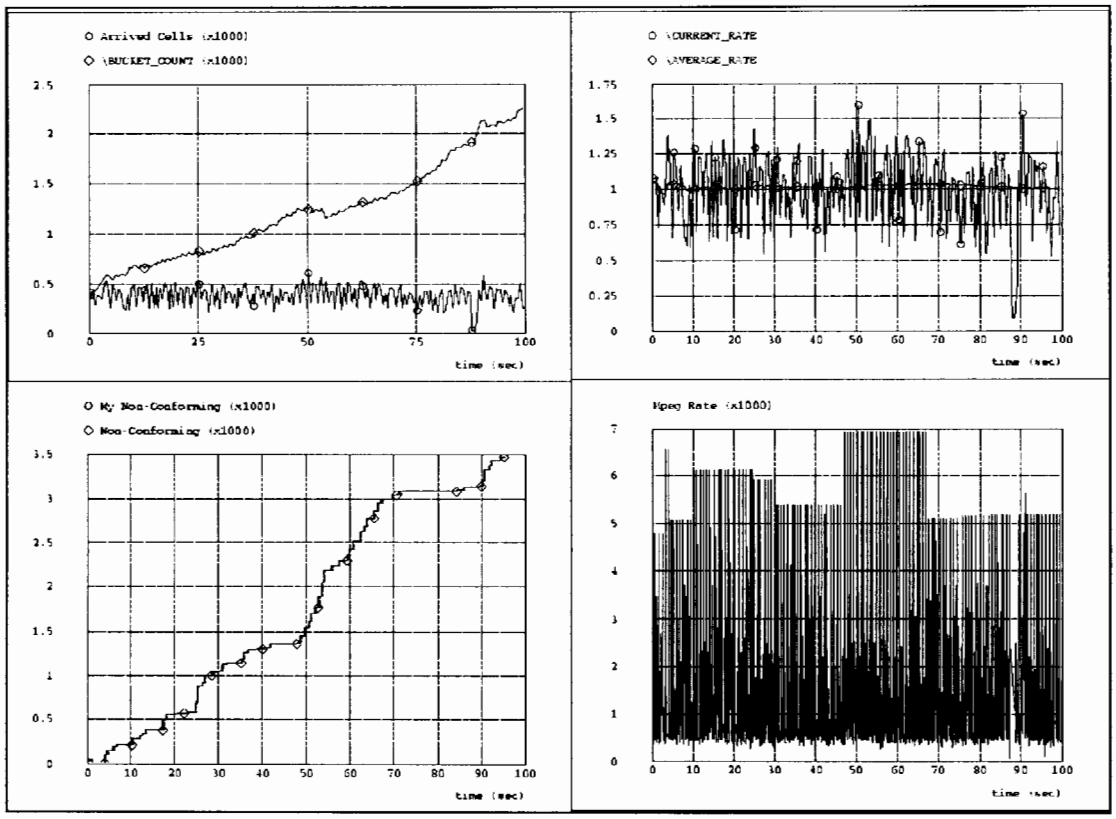



Figure 6-3 Conforming source - seed = 300, m = 1.

Figure 6-3 shows a source that is conforming. Its average cell rate is equal to its negotiated average cell rate at the end of the simulation. From this graph one sees that the GCRA drops more cells than the modified Fuzzy Logic Congestion Controller. In fact the GCRA drops more than 3500 cells while the Fuzzy Logic Congestion Controller drops less than 100. From the number of cells that arrived at the start of the simulation one can see that the IBS of the Fuzzy Logic Congestion Controller was too small. This is what caused some of those initial cells to be dropped. As time increased the amount of credit that the source owned also increased, and no further cells were dropped. The amount of credit is shown in Figure 6-3 by the BUCKET COUNT statistic.

One finds that the other nine simulations for the conforming source, all show the same trend: the GCRA drops more cells than the modified Fuzzy Logic Congestion Controller.

### 6.3.2 Case 2 - Test 1 - Non-conforming Source

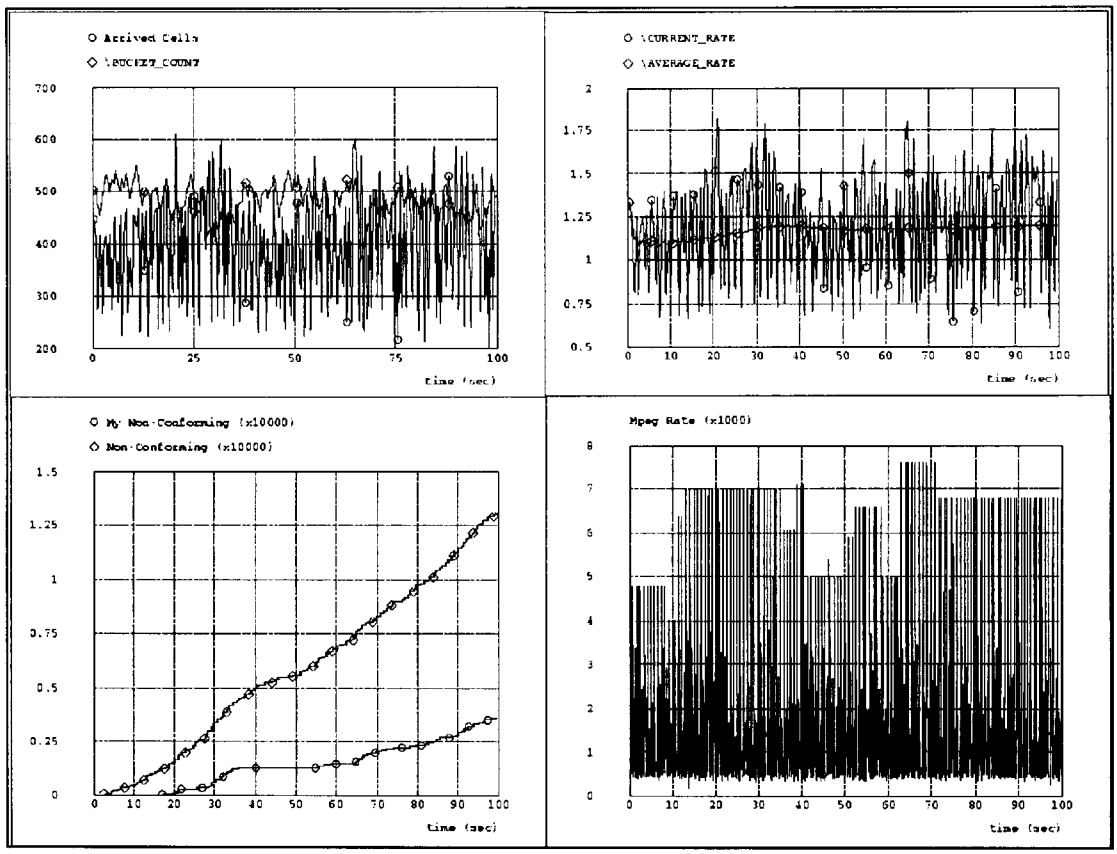



Figure 6-4 Non-conforming source, seed = 600, m = 1.2.

Figure 6-4 shows the case where the source is non-conforming and the average cell rate at the end of the simulation is 1.2 times the negotiated average cell rate.

In this case one sees that the Fuzzy Logic Congestion Controller only starts dropping cells after about 15 seconds into the simulation (the My Non-conforming statistic). This happens when the number of cells that arrived during the window period is greater than the amount that was expected. The credit level no longer increases, as was the case with the conforming source, and actually drops slightly as time passes.

The GCRA again drops many more cells than the Fuzzy Logic Congestion Controller. The Fuzzy Logic Congestion Controller is also starting to look like a scaled down version of the Non-conforming statistic of the GCRA. This shows that the controller is close to performing like the GCRA when policing non-conforming sources. This trend is emphasised in the next test.

### 6.3.3 Case 2 - Test 2 - Non-conforming Source

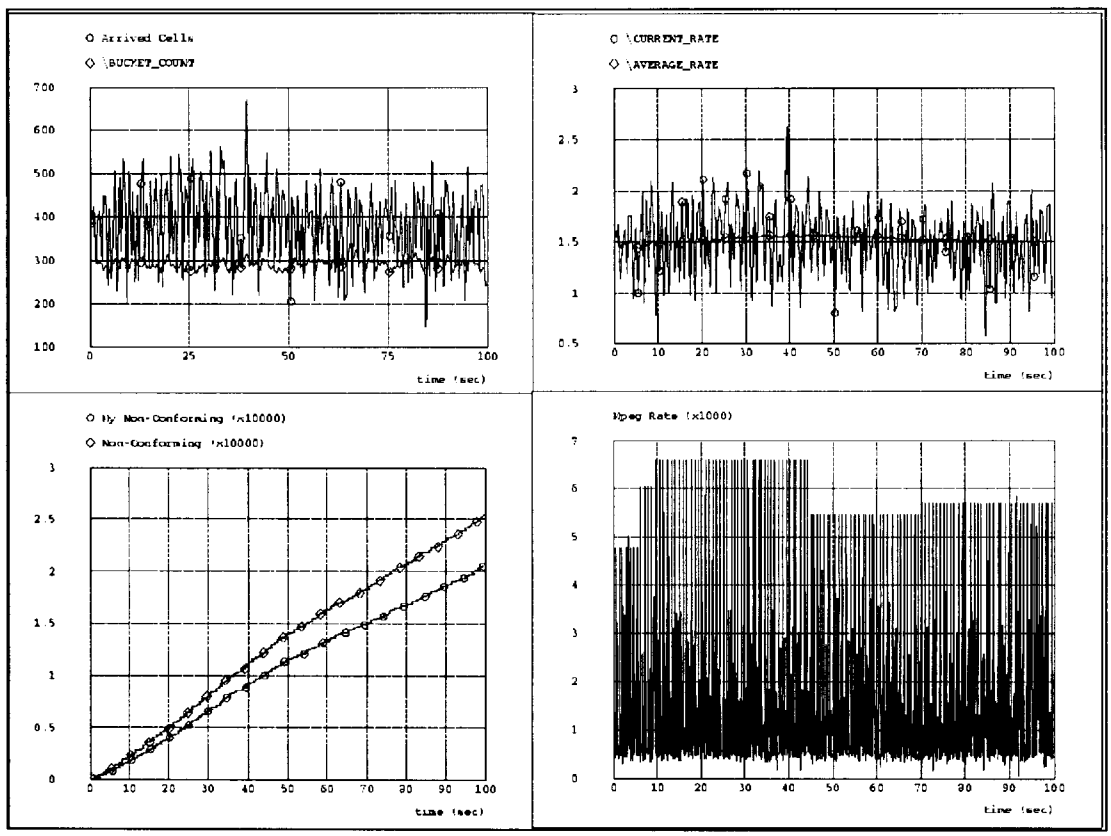



Figure 6-5 Non-conforming source, seed = 500, m = 1.5.

Figure 6-5 shows a source that is extremely non-conforming and has an overload factor m of 1.5.

The graphs show that the number of cells that arrive at the controller is in most cases much more than the expected value.

Here one sees the number of cells that the two methods drop, and the rate at which this happens, is very close. This further emphasises the trend that the Fuzzy Logic Congestion Controller tends towards the GCRA when the source is non-conforming.

## 6.4 In Summary

The results presented in the previous three examples are similar to the other 27 simulation results that are presented in Appendix C. They can thus be considered as a valid representation of the general trend of how the two control methods perform. Comparing the different simulations, one sees that GCRA tends to drop more cells than the Fuzzy Logic Congestion Controller.

In the case where the source is conforming this difference is very pronounced. The Fuzzy Logic Congestion Controller only tends to drop cells if the Initial Burst Size that was selected, was too low. But as time passes, the credit level of the source is increased so that no further cells are dropped.

In the cases where the source is non-conforming, one finds that the performance of the modified Fuzzy Logic Congestion Controller tends toward that of the GCRA as the level of non-conformity increases.

When considering how the Fuzzy Logic Congestion Controller polices the MPEG source, one also finds that it presents a greater probability that the I frames of the compression sequence will be allowed through unmodified. This happens because the Fuzzy Logic Congestion Controller updates it credit level at the start of each new window period, and is more likely to allow the most important frame, the I frame, through. This is not the case with the GCRA.

The simulations show that the Fuzzy Logic Congestion Controller has merit. The modifications that were made to the previous controller can thus be justified when considering the improvement in performance that was obtained.

# 7. CONCLUSIONS AND RECOMMENDATIONS

A Fuzzy Logic Congestion Controller was designed to prevent congestion in ATM networks. It allows conforming cells through to the network, while tagging or dropping non-conforming cells. However, when compared with the performance of the GCRA, it did not perform as well. It was found that in extreme cases of non-conformity the fuzzy logic controller dropped all cells from the input stream, while the GCRA allowed the negotiated amount of cells though and dropped the rest.

The controller was then modified to see if it could equal or better the performance of the GCRA. Simulations were run on a MPEG-encoded VBR source to determine how well the two control methods policed a VBR source that had practical application in the real world.

The simulations showed that in the case where the source was conforming to its traffic contract, the GCRA dropped many more cells than the Modified Fuzzy Logic Congestion Controller (MFLCC). It was found that the MFLCC only dropped cells at the start of the connection if one of its input parameters (the initial burst size) was too low. However, as time passes, the controller determines that the source is conforming and modifies its control action accordingly, allowing all cells through. The GCRA, on the other hand, drops cells throughout the duration of the connection. This is particularly the case if the ratio of the peak cell rate to the average cell rate is high and the algorithm is used to police the sustainable cell rate. This shows that it is difficult for the GCRA to police a statistical value such as the mean cell rate. Thus, the MFLCC performs better than the GCRA when policing a conforming MPEG-encoded VBR source.

The simulations showed that the control action of the MFLCC tends towards that of the GCRA as the level of non-conformity increases. As the level increases the controller makes sure that only the accepted number of cells is allowed through. This means that the MFLCC applies a stricter level of control as the source becomes increasingly non-conforming. It was also found that the MFLCC performs better than the FLCC when this happens because the MFLCC still allows cells through where the FLCC did not.

The MFLCC is especially well suited for sources where a window period can be defined where the most important information is presented at the start of the windowing period. The MFLCC updates its credit level at the beginning of each window period, i.e. the amount of credit available to the source is at a maximum at the start of the window. Cells that thus arrive at that time have the greatest probability of being allowed through to the network. One can also think of this set-up as assigning a "higher" priority to cells arriving early in the window period.

MPEG sources are good examples of this kind of traffic sources: it has a well-defined traffic pattern (a set number of frames per Group of Pictures) and the highest priority data occur at the start of the GOP. It is thus easy to define the window period as the time it takes to transmit the GOP. This means that the I-frames are sent at the start of the window when the credit is at its maximum, making sure that the most important frame has the highest probability of being allowed through without one of its cells being dropped. The GCRA has no mechanism for coping with this: there is no guarantee that its credit level will be at a maximum when the I-frame arrives. One thus finds that the MFLCC performs better than the GCRA when policing sources such as MPEG traffic.

An alternate method is needed to determine the window period if it is not obvious. This can be the case, for example, when the controller has to police a traffic source with a gaussian distributed arrival rate. In this case there is no repetitive pattern to take advantage of (as was the case with MPEG's GOP). This alternate method would ultimately have to consider the characteristics of the source in order to make its decision. The simulations that were run on the MFLCC concentrated on an MPEG source, and thus did not consider this aspect of the design. It is recommended that the formulation of this alternate method be considered in any future work based on this dissertation.

The field of traffic control in ATM is still wide open. New control methods are surfacing constantly, ranging from mathematically intensive approaches to the use of Artificial Intelligence. It is hoped that this dissertation will help to highlight the practicality and validity of using AI, and especially Fuzzy Logic, in this new and ever evolving technological environment.

# REFERENCES


[ 1 ] R. Warfield, S. Chan, A. Konheim and A. Guillaume, "Real-time traffic estimation in ATM networks," ITC 14, 1994.

[ 2 ] E. Rathgeb, "Modeling and performance comparisons of policing mechanisms for ATM networks," IEEE Journal of Selected Areas in Communication, vol. 9, no. 3, pp. 325-334, April 1991.

[ 3 ] F. Denissen, E. Desmet, and G. H. Petit, "The policing function in an ATM network,", Proceedings of the 1990 International Zurich Seminar in Digital Communication, March 1990.

[ 4 ] Professor M. Braae, Control Engineering course notes for EEE330S (1993) and EEE417F (1994), University of Cape Town, Dept. of Electrical Engineering.

[ 5 ] Eckberg et al., "Controlling congestion in B-ISDN/ATM: Issues and Strategies.", IEEE Communications Magazine, September 1991.

[ 6 ] W. H. Tranter and Kurt L. Kosbar, "Simulation of Communications Systems," IEEE Communications Magazine, July 1994.

[ 7 ] Koralov et al., "Application of Kalman Filter in high speed networks," IEEE Globecom 1994.

[ 8 ] J. Giarratano and G. Riley, Expert Systems: Principles and Programming, PWS-KENT Publishing Company.

[ 9 ] Thomas D. Ndousse, "Fuzzy Neural control of voice cells in ATM networks," IEEE Journal on Selected Areas in Communications, vol. 12, no. 9, December 1994.

[ 10 ] Vincent Catania, Guiseppe Ficili, Sergio Palazzo, and Daniela Panno, "A comparative analysis of Fuzzy versus Conventional Policing mechanisms for ATM networks," IEEE/ACM Transactions on Networking, vol. 4, no. 3, June 1996.

[ 11 ] Ray-Guang Cheng and Chung-Ju Chang, "Design of a Fuzzy Traffic Controller for ATM networks," IEEE/ACM Transactions on Networking, vol. 4, no. 3, June 1996.

[ 12 ] C. H. Chen, Fuzzy Logic and Neural Network Handbook, ISBN 0-07-011189-8, 1996.

[ 13 ] Jean-Yves Le Boudec, "The Asynchronous Transfer Mode: A tutorial", Computer Networks and ISDN Systems 24 (1992), pages 278-309, North Holland.

[ 14 ] F. M. Albrecht, "Using Fuzzy Logic for Congestion control at the UNI," Regional International Teletraffic Conference, South Africa,1995, pg. 352-358.

[ 15] Detlef Jensen, 'B-ISDN Network Management By A Fuzzy Logic Controller,' IEEE Globecom '94.

[ 16 ] C. Chang and R. Cheng, 'Traffic Control In An ATM Network Using Fuzzy Set Theory,' Proceedings of INFOCOM '94.

[ 17 ] S. Yazdi and H. T. Mouftal, "Congestion control methods for B-ISDN," IEEE Communications Magzine, vol. 30, no. 7, pages 42-47, July 1992.

[ 18 ] The ATM Forum, ATM User-Network Interface Specification, Version 3.0, published by PTR Prentice Hall, ISBN 0-13-225863-3, 1993.

[ 19 ] Marwan Krunz and Herman Hughes, "A Traffic Model for MPEG-Coded VBR Streams", source unknown. The authors can be contacted at the Department of Electrical Engineering and the Department of Computer Science respectively, at the Michigan State University, East Lansing, MI 48824. E-MAIL [krunz, hughes]@cps.msu.edu.

[ 20 ] D. Le Gall, "MPEG: A video compression standard for multimedia applications," Communications of the ACM, Vol. 34, April 1991.

[ 21 ] Catania et al., "Using Fuzzy logic in ATM Source Traffic Control," IEEE Communications Magazine, November 1996, Vol. 34, No. 11.

[ 22 ] William Stallings, ISDN and Broadband ISDN with Frame Relay and ATM, Prentice Hall, ISBN 0-02-415513-6.

[ 23 ] Kerry W. Fendick, "Evolution of Controls for the Available Bit Rate Service," IEEE Communications Magazine, Nov. 1996, Vol. 34, No. 11, pg. 35 - 39.

[ 24 ] H. Saito et al., "Performance Issues in Public ABR Service," IEEE Communications Magazine, Nov. 1996, Vol. 34, No. 11, pg. 40 - 48.

[ 25 ] R. Jain et a., "Source Behaviour for ATM ABR Traffic Management: An Explanation," IEEE Communications Magazine, Nov. 1996, Vol. 34, No. 11, pg. 50 - 57.

[ 26 ] E. Nordstrom et al., "Neural Networks for Adaptive Traffic Control in ATM Networks," IEEE Communications Magazine, October 1995, Vol. 33, No. 10, pg. 43 - 49.

[ 27 ] J. E. Neves et al., "Neural Networks in B-ISDN Flow Control: ATM Traffic Prediction or Network Modeling?" IEEE Communications Magazine, October 1995, Vol. 33, No. 10, pg. 50 - 56.

[ 28 ] Atsushi Hiramatsu, "Training Techniques for Neural Network Applications in ATM," IEEE Communications Magazine, October 1995, Vol. 33, No. 10, pg. 58 - 67.

[ 29 ] Young-Keun Park and Gyungho Lee, "Applications of Neural Networks in High-Speed Communication Networks," IEEE Communications Magazine, October 1995, Vol. 33, No. 10, pg. 68 - 74.

[ 30 ] A. A. Tarraf et al., "Intelligent Traffic Control for ATM Broadband Networks," IEEE Communications Magazine, October 1995, Vol. 33, No. 10, pg. 76 - 82.

[ 31 ] Giovanni Gallus, "Load Forecasting in Integrated Services Digital Network (ISDN) carrying voice and non voice traffic," Regional International Teletraffic Seminar, South Africa, 1995, pg. 661 - 670.

[ 32 ] M. F. Scheffer et al., "Improved Modelling Techniques for Packetized ISDN," Regional International Teletraffic Seminar, South Africa, 1995, pg. 672 - 682.

[ 33 ] Fuzzy Clips Website: http://ai.iit.nrc.ca/fuzzy/fuzzy.html.

# BIBLIOGRAPHY


1. The ATM Forum, **ATM User-Network Interface Specification Version 3.0**, PTR Prentice Hall, Englewood Cliffs, New Jersey 07632.

2. Chen, C. H., **Fuzzy Logic and Neural Network Handbook**, McGraw-Hill, ISBN 0-07011189-8 (hb).

3. De Prycker, Martin, **Asynchronous Transfer Mode**, Second Edition, Ellis Horwood, ISBN 0-13-178542-7.

4. Stallings, William, **ISDN and Broadband ISDN with Frame Relay and ATM**, Prentice-Hall, Englewood Cliffs, NJ 07632, ISBN 0-02-415513-6.

5. Tanenbaum, Andrew S., **Computer Networks**, Third Edition, Prentice-Hall, ISBN 0-13-394248-1

# APPENDIX A

## AN INFORMAL FUZZY LOGIC PRIMER

This appendix presents an introduction to the concepts behind fuzzy logic in an informal, conversational manner. Many texts provide a different approach - a more formal one - and are often long winded as a result. The informal approach was chosen here to ease the learning curve of getting to grips with Fuzzy Logic, keeping things as simple as possible for the novice, while giving a good working knowledge of the important concepts behind designing with Fuzzy Logic.

This document resulted after someone approached the author after a conference in 1995, asking how fuzzy logic worked. The conversations were done via e-mail, and an informal and relaxed relationship developed between the two. It was then decided that a short informal explanation, developed around an example showing the key principles involved, would be best to supplement the discussions. This appendix contains that short informal tutorial that was written one and a half years ago.

Thank you, Marius.

Hi, Marius.

In this document I'll explain the processes involved with fuzzy logic. I thought up a very simple control problem just to show how the fuzzification, inference, rule application and defuzzyfication processes work. If you have any questions then don't hesitate to call me.

Here goes the simplistic example. Consider an inclined plane at an angle of 30 (theta) degrees to the horizontal. Make this plane say 30 cm long (taking the bottom of the plane being at 0 cm while the end is at 30 cm). Now imagine there's a groove running down the middle of this plane so that when a marble is placed on/in it, the marble can role down this groove taking the shortest possible route downwards. Note that the marble will accelerate downwards due to gravity, with the acceleration being,

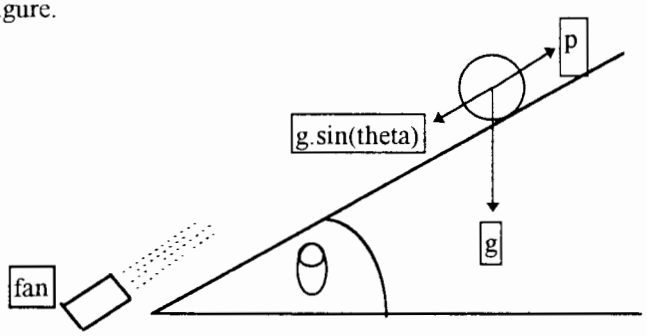
$$a=g.\sin(\text{theta})$$

with theta=30°,

$$g=10\ m/s^2.$$

But, to make things interesting, lets put a fan at the bottom of the plane to blow the marble back up the plane. For simplicity's sake (bear with me) lets assume that all the air coming from the fan goes directly at this marble and effectively provides an upwards acceleration of p $m/s^2$, and we can control how hard this fan blows (i.e. we can control p). P >= 0.

Here's the figure.




Now, the object of this control problem is to keep the marble at a certain point on this plane by using the fan.. Lets make the position of the marble x, and the velocity v, defining v positive if its going upwards, and negative if downwards. Going back to basic physics,

$$v=u+at, \text{ and}$$

$$s=s_0 + ut + 0.5at^2.$$

Since we'll be using a fuzzy controller we'll have a sampled system, so our t will be a $\Delta t$. Lets make $\Delta t$=5 ms.

In this control problem we've got two inputs, x and v, and one output p, and lets make our objective to keep the marble at x=15.

The fuzzy system is designed as follows:

Step 1: Fuzzification.

To do this the variables in our system has to be described in words by means of fuzzy sets. Position x I'm going to define as follows:

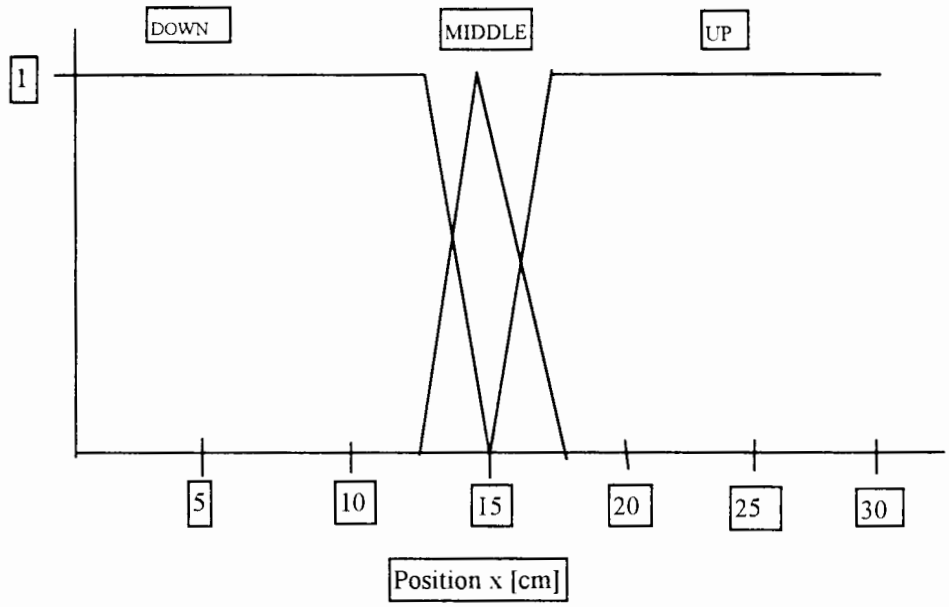



Figure 1. Sets for position x.

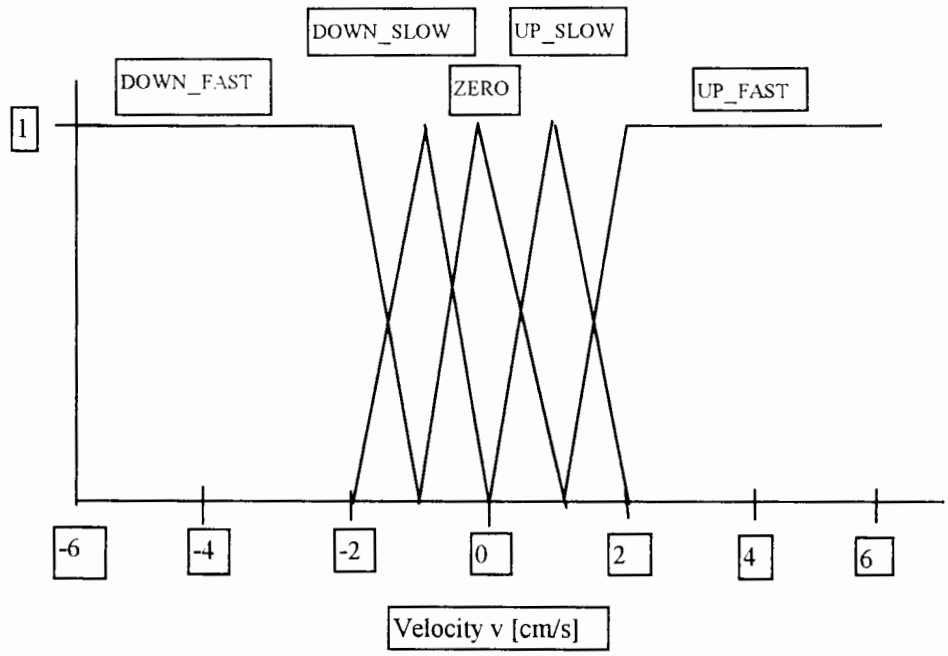



Figure 2. Sets for velocity v.

From figure 1 x is described as being either DOWN, MIDDLE or UP, while figure 2 shows the velocity to be either DOWN_FAST, DOWN_SLOW, ZERO, UP_SLOW, or UP_FAST. So if x=16.25 then x can be described as being 25% UP or as a membership of .25 of being UP, as well as having a membership of .75 for MIDDLE.
If v=4 then it has a membership of 1 to UP_FAST, while having a membership of 0 for all the other sets. This process of 'mapping' the crisp (real life) value over to a linguistic value (a word) and assigning a membership value to it, is called FUZZIFICATION.

Lets define a few rules of that our controller must implement/apply.

Step 2. Defining the rules.

If you have only two variables then an easy way of setting up your rule base is in the form of a two dimensional table, where the cells in the table represent the control action to be taken when certain combinations of input variables occur. In order to say what the control action must be, we must first define what the possible range of output action 'states' is. We must therefore define our fuzzy sets for the output variable p.

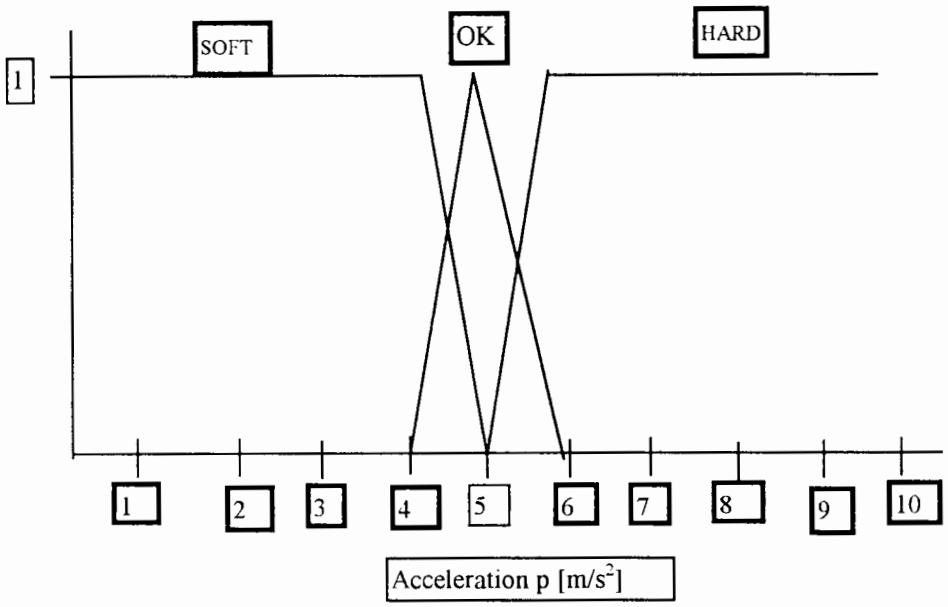



Figure 3. Acceleration p fuzzy sets.

OK, we can now define our rules.

Table 1: Rows x, columns v.

| | FAST_DOWN | SLOW_DOWN | ZERO | SLOW_UP | FAST_UP |
|---|---|---|---|---|---|
| DOWN | HARD | HARD | HARD | OK | SOFTLY |
| MIDDLE | HARD | HARD | OK | SOFTLY | SOFTLY |
| UP | HARD | OK | SOFTLY | SOFTLY | SOFTLY |

This says that if for example the ball comes SLOWLY DOWN and the marble is DOWN then blow HARD, also if the marble is going SLOWLY UP and is in the MIDDLE of the slope then blow SOFTLY, etc. These rules are fairly simple, but it shows the principle of how it is done.

Step 3. Now that the rules are defined, lets do the INFERENCE process.

Remember, before I had x=16.25 and v=4, which meant that x was 0.75 MIDDLE and .25 UP, while v was 1 FAST_UP. We use table 1 to see what control action should be taken. Two rules fire:1) x=MIDDLE, v=FAST_UP fires p=SOFTLY, and 2) x=UP, v=FAST_UP fires SOFTLY. (If x was MIDDLE and v was SLOW_DOWN then an additional fire would have been HARD. Check the table.)

So how do we decide which output to use if there is more than one output (remember TWO rules fired). We combine the outputs as follows. Take the output of each rule that fired and mark its relevant set in the output variable' s sets as being active. Now clip that set at the minimum value of the member set values of its two input sets. (Wow, that confuses even me!). E.g. For rule 1 x's membership was .75, and v's was 1. The MIN of these two memberships is .75, so clip the set at .75 (if they were the same, say membership x=.8, membership v=.8 then the min would have been .8 and we would have to clip at .8). Figure 4 shows this.

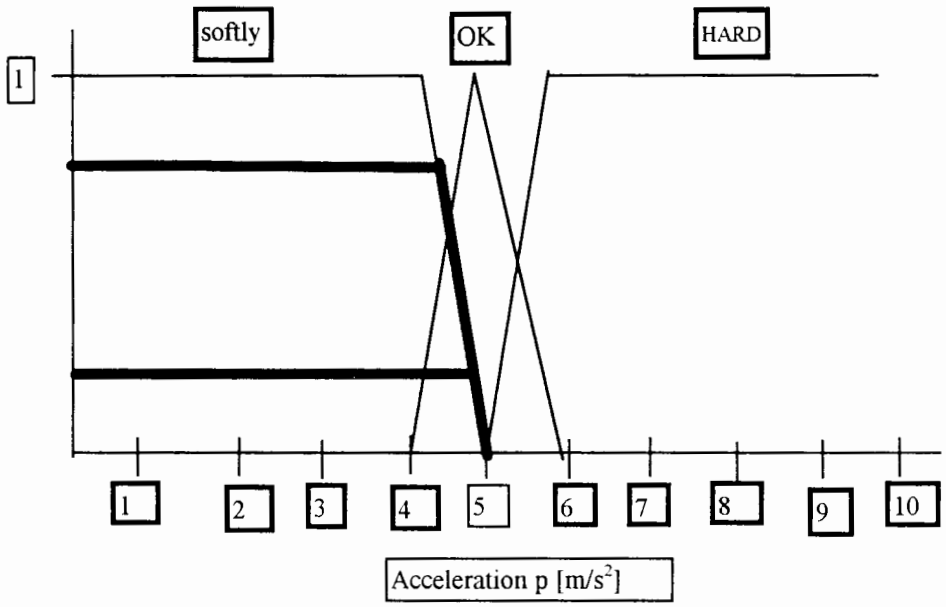



Figure 4. Clipping softly at .5.

Follow the same procedure for rule 2: membership x=.25, membership v=1, MIN is .25, so clip softly at .25 resulting in the bottom thick line in figure 4.

Now that all the rules have been handled, take the MAXimum of all the sets. Note that the sets that were not selected are clipped at 0. This gives figure 5.

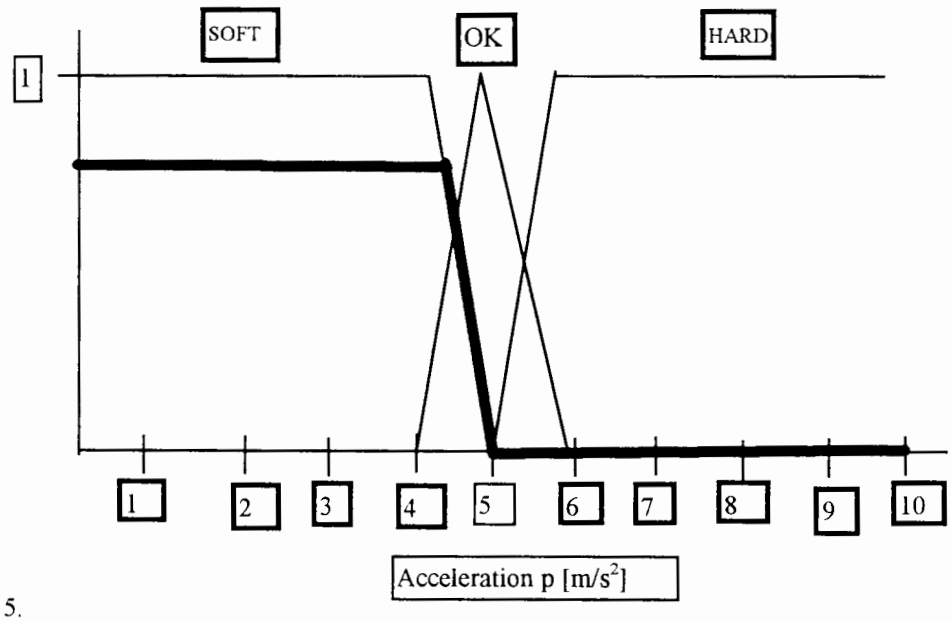



Figure 5.

Aside:
Just say for instance if another set was activated, take OK for example and the min of its two input sets' membership was .5 then figure 4 would have looked like figure 6 and figure 5 like figure 7.

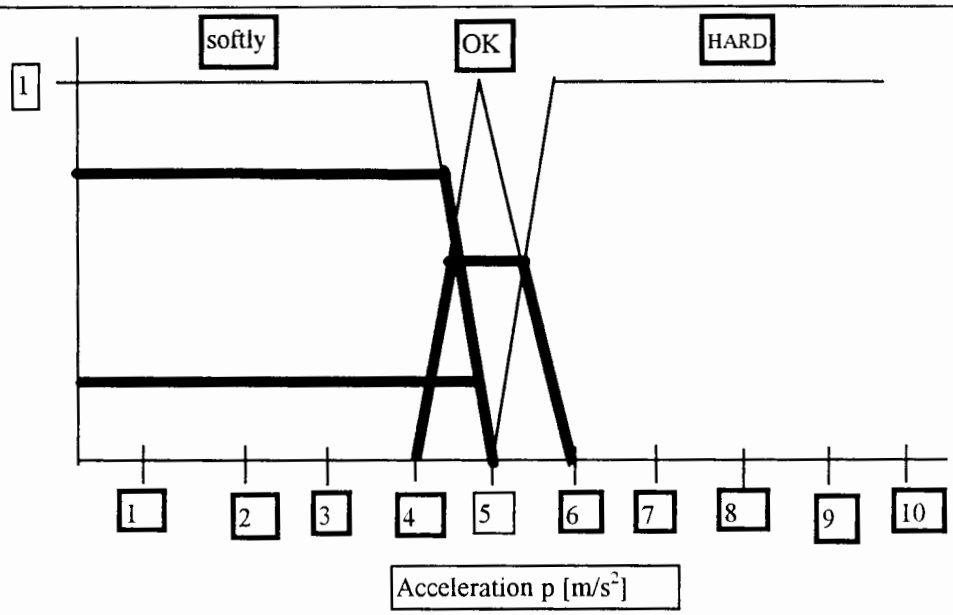



Figure 6.

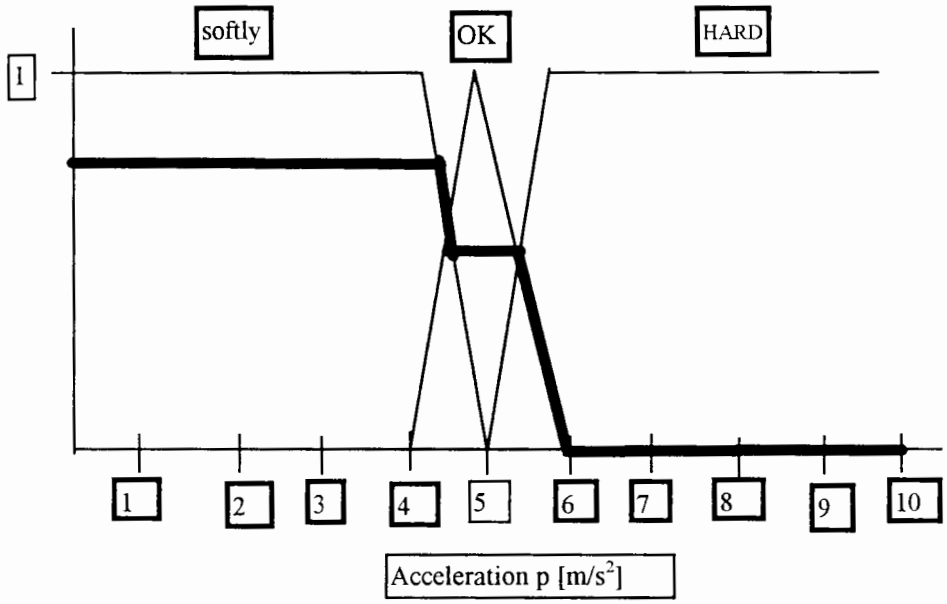



Figure 7.

End of Aside.

Here I showed you one method to do the inference process, and that is the MAX-MIN inference process. There are various other methods, but this seems to work well enough.

OK, so now we've got this weird looking fuzzy set, how does one get the actual crisp output value out of this. That gets us to step 4.

Step 4. The DEFUZZIFICATION process.

The crisp value is obtained by calculating the centroid of that weird set and using its x co-ordinate as the crisp output. (The x co-ordinate of the centroid has some name, but I can't get to it right now). For figure 5 the output would then be about 2.5 while for figure 7 it would be about 3. As in the case of the inference process there is more than one way of doing the defuzzyfication process, but in most cases the centroid works the best. So, for figure 5 p=2.5, and for figure 7 p=3.

That's it! You now know it all! If I have not made myself clear enough in certain areas then please tell me so. I would like to get some feedback on this document.

See you.

Fred

# APPENDIX B

## FUZ - THE FUZZY LOGIC ENGINE

Fuz is the fuzzy logic engine that was developed during the course of this thesis. It allows the user to define the input and output fuzzy variables, as well as the fuzzy rule base that holds the knowledge base of the system. Once the engine is initialised, it fuzzifies the input variables, applies the rule base and produces the output value(s).

Fuz started off as a DOS based standalone fuzzy logic engine. The DOS version had an interactive command environment where the engine executed the commands the user entered at the command prompt. It allowed the user to define fuzzy sets and rules at the command prompt, and to display the values of selected variables in a graphical environment. A mathematical expression evaluator was also included to simplify the use of equations in the fuzzy engine, and variables (fuzzy and normal) that were defined in Fuz could be saved to disk and reloaded at a later time.

The DOS version of Fuz was however not compatible with OPNET®, which runs under a UNIX environment. Particular points of concern were:

1. How to make use of OPNET® declared variables, inside Fuz.
2. Converting the interactive aspect of the DOS Fuz to work in OPNET®'s mostly non-interactive simulations.
3. Converting the DOS based graphics environment to UNIX.
4. How to make use of OPNET®'s graphical simulation results capabilities.

Fuz was ultimately designed to enable the author to use fuzzy logic inside OPNET®, thus it was decided to convert the DOS based application to a UNIX based one. This involved discarding the DOS graphical system and changing to a non-interactive environment. The resulting fuzzy logic engine had a very close interaction with OPNET®.

Fuz was extended to "share" variables that were defined in OPNET®. This was done by allowing OPNET® to register its variables with Fuz, and variables defined in Fuz were made available to OPNET® in a similar manner. This meant that data collected from, for example, a congestion controller being simulated in OPNET®, could be used in Fuz where fuzzy logic methods could be applied on it.

The UNIX version of Fuz is non-interactive, and receives its instructions from a file called the input file. This input file typically instructs Fuz to reset itself, load another file, called the fuzzy file, in which the fuzzy variables and rules are defined, and then initialise the loaded file. The user thus no longer had to interactively, i.e. manually, configure Fuz for each simulation, but merely had to define the instructions in an input file that could be used repeatedly. This meant that simulations could now be done without any user interaction.

Fuz was also extended to produce graphs in a format that OPNET® could use. An added feature was that the fuzzy file could contain instructions for the fuzzy logic engine to produce statistics in the format that OPNET® could use for its graphs. Effectively this meant that the fuzzy file could be used to "plot" the values used in Fuz to statistics that were compatible with OPNET®. These "plotted" values could then be displayed using OPNET®'s normal capabilities for graphically displaying simulation results. Since OPNET® variables that were registered with Fuz were also available to the fuzzy logic engine, it was also possible to plot OPNET® variables *outside*[1] of the simulation environment. The number and selection of statistics can thus be decided upon *after* the simulation has been compiled to an executable file.

The rest of this appendix presents the format of the fuzzy file, and how to integrate Fuz into OPNET®.

---

1 OPNET® files are compiled into a single executable file, which when run produces the statistics that the user specified in the simulation environment. There is no way to change what statistics have to be plotted once the executable file has been compiled (without actually hacking into the program, or using a probe file). The fuzzy file could however contain instructions to produce any statistic or variable to which it has access. The executable file does thus not need to be recompiled when, for example, a new statistic is required.

## 1. Formatting the Fuzzy File

Describing the format of the fuzzy file includes how to:

1. define fuzzy variables,
2. define fuzzy rules,
3. define and use local and global Fuz variables,
4. plot statistics, and
5. define comments.

### 1.1 Defining Fuzzy Variables

A fuzzy variable is used to group fuzzy sets belonging to the same variable, and has to contain at least one fuzzy set. It also stores the range over which the variable is defined, as well as the initial value that will be assigned to the variable.

Fuzzy variables have the following format:

```
var         fuzzy_variable_name     [input | output]                          (1)
            init          initial_value                                       (2)
            min_value     max_value                                           (3)

            fuzzy_set_name< val prob > < val prob > < val prob > < val prob > | (4)
            fuzzy_set_name< val prob > < val prob > < val prob > <  >          (5)
            {. . .}                                                           (6)
endvar                                                                        (7)
```

Comments:

- Words in bold are tokens that form part of the Fuz's language and may not be used as variable names.
- Words in italics are any C-compatible variable name.
- Any other words are floating point values consisting of the digits 0 to 9 and the decimal point.
- Note the spaces before and after each angular bracket (< and >).
- The number at the end of each line indicates the line number and does not form part of the fuzzy variable structure. Lines will be referred to by those numbers in the text that follows.
- {. . .} means that more fuzzy sets may follow.

Line 1 defines a new fuzzy variable as either an input or an output variable.

Line 2 sets the value to which the fuzzy variable will be initialized at the start of the simulation.

Line 3 defines the range of values over which the fuzzy variable is valid. The two values specify the minimum and maximum values of the range.

Lines 1 to 3 must always follow in that order.

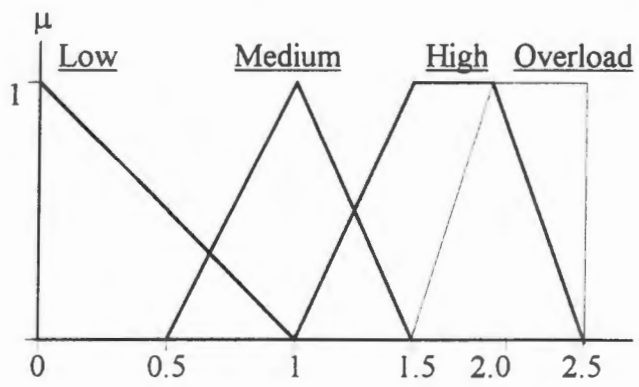
Lines 4 and 5 are used to define fuzzy sets. Each line starts with the name for the fuzzy set. The next four coordinates (the two values between the open and close angular brackets) define the shape of the fuzzy set. A triangular set requires three points, a trapezoidal set requires four points.

The fourth coordinate may be left out, but the brackets must remain (line 4). If this is the case then internally the fourth coordinate is set to the values specified for the third coordinate.

Line 7 indicates the end of the fuzzy variable declaration.

An example fuzzy variable and its corresponding fuzzy file description follow:

The Current Rate
vs. its Membership function



```
var   current_rate input
      init   0.0
      0      2.5

      low          < 0 0 >    < 0 1 >    < 1 0 >    < >
      medium       < 0.5 0 >  < 1 1 >    < 1.5 0 >  < >
      high         < 1 0      < 1.5 1 >  < 2.0 1 >  < 2.5 0 >
      overload     < 1.5 0 >  < 2.0 1 >  < 2.5 1 >  < >
endvar
```

### 1.2 Defining Fuzzy Rules

Each rule consists of one or more IF statements and a number of THEN statements. If all the IF statements of the rule evaluate to TRUE then the THEN statements are processed.

A fuzzy rule has the following format:

```
rule          fuzzy_rule_name                                  (1)
              fuzzy_variable_name      fuzzy_set_name          (2)
              {. . .}                                          (3)
->                                                             (4)
              fuzzy_variable_name      fuzzy_set_name          (5)
              {. . .}                                          (6)
              eqn          y=cos(x)                            (7)
              {. . .}                                          (8)
endrule                                                        (9)
```

Line 1 defines a new fuzzy rule.

Line 2 shows an IF statement. It contains a predefined fuzzy variable and one of its fuzzy sets. If the fuzzy set of that fuzzy variable is active, then the IF statement evaluates to TRUE.

Line 3 shows that more than one IF statement may be defined.

Only input fuzzy variables may be used in IF statements.

The arrow sign ( -> ) in line 4 determine the end of the IF statements and the start of the THEN statements.

Only output fuzzy variables may be used in THEN statements.

If all the IF statements are TRUE, then the fuzzy sets of the corresponding output fuzzy variables are set active.

Fuzzy rules may also contain equations. Equations will be handled in section 1.3.

An example fuzzy rule base and its corresponding fuzzy file description follow:

| BUCKET / CCR | LOW | MEDIUM | HIGH |
|---|---|---|---|
| LOW | PB | PB | PB |
| SPOT-ON | PB | PS | |
| HIGH | PB | | Z |

Table 1. Input fuzzy variables are Bucket and CCR, output fuzzy variable is SA.

| BUCKET / CCR | LOW | MEDIUM | HIGH |
|---|---|---|---|
| LOW | | | |
| SPOT-ON | | PS | PB |
| HIGH | | NB | PS |

Table 2. Input fuzzy variables are Bucket and CCR, output fuzzy variable is Out.

```
rule   r1
       bucket      low
->
       sa          pb
endrule

rule   r2
       ccr         low
->
       sa          pb
endrule

rule r3
       bucket      medium
       ccr         spot-on
->
       sa          ps
```

### 1.3 Defining and Using Local and Global Fuz variables,

Fuz includes a mathematical expression evaluator[2] (EE). It can evaluate expressions of the following form:

```
"    1+1
     10 * (x=5)            <== Assigns 5 to X first!
     ((1/3) * sin(45))^2
     X=50
     Y=100
     z=hypot(x,y)
```

As you can see, it supports variable assignments and referencing, as well as functions. Variable and function names are limited to 16 characters in length, but this is easily modified in the .H file. Functions may have anywhere from 1 to 3 arguments, again this is modifiable.

It also maintains a set of "read-only variables" (constants) that can be referenced just like variables, but they cannot be permanently altered by the user."[3]

EE contains the following:

1. mathematical functions:
   sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, exp, log, log10, sqrt, floor, ceil, abs, (single parameter) and hypot (two parameters),
2. conversion functions:
   deg (converts radians to degrees), and rad (converts degrees to radians),
3. comparative functions:
   equal, bigger, and smaller (comparing two parameters, returns 1 if true, 0 if false),
   not (returns 1 if parameter is not 1, else returns 0),
4. a random number function:
   random (returns a random number less than the single parameter),
5. constants:
   PI (3.14159265358979323846) and
   E (2.71828182845904523536).

Global and local variables may also be used to store values. Global variables are defined outside fuzzy variable, rule and plot structures, while local variables are defined in the

---

[2] Based on the mathematical expression evaluator provided by Mark Morley, email morley@camosun.bc.ca, and can be found on the SIMTEL depository as C_EVAL.ZIP under the C sub-directory.

[3] Taken from Mark Morley's document file.

THEN statement section of fuzzy rules. These variables may only be used after a value has already been assigned to them.

A variable is defined by assigning a value to it in an equation. For example

```
eqn     x=sin(PI/2)
```

sets x equal to 1. If x has been assigned previously then the old value is overwritten. Once the variable has been defined it may be used in other equations, for example

```
eqn     y=3*x+2
```

sets y equal to 5.

Variables that are defined in the THEN section of rules are referred to as local variables, and are deleted once the rule has been processed. For example,

```
eqn     x=1                                     (1)

rule    dummy_rule                              (2)
        ccr     high                            (3)
->                                              (4)
        sa      pb                              (5)
        eqn     y=x                             (6)
        eqn     z=y+x                           (7)
        out     ps                              (8)
endrule                                         (9)

eqn     x=2+y                                   (10)
eqn     x=2*x                                   (11)
```

Line 1 declares a global variable, x, while lines 6 and 7 declare two variables, y and z, that are local to rule dummy_rule. Local variables may be used inside a rule, but are deleted once Fuz exists the rule. That is why line 10 is not valid (and will cause an expression error) while line 11 is valid.

Local variables may also have the same name as global variables, and when used in equations the ~ is used to differentiate between the two types of variables to which access is required. For example,

```
eqn     w=3                 // global w=3
eqn     x=1                 // global x=1

rule    test                // new rule
        ccr     high        // if . . .
->                          // then . . .
        eqn     x=3         // local x=3
        eqn     y=x         // local y=local x=3
        eqn     z=~x        // local z=global x=1
        eqn     w=y+z       // local w=local y+local z=4
        eqn     ~x=0        // global x=0
endrule                     // local w, x, y and z are destroyed

eqn     w=w+1               // global w=4
eqn     x=x+1               // global x=1
```

Comment:

- The //<space> denotes a comment

Equations are useful if one needs to simulate an environment inside Fuz. For example, the example presented in Appendix A can be formulated in Fuz, as a whole, and can be simulated by calling the fuzzyfication and defuzzyfication processes repeatedly. This example can be found on the disk that accompanies this thesis. The file is called MARBLE.FUZ.

### 1.4 Defining Comments

A comment is any text following two forward slashes and a space, that is //<space. For example.

```
// this is a comment
```

```
eqn        a=b                          // this is another comment
```

Fuz ignores everything between the //<space> and the end of the line.

### 1.5 Plotting Statistics

The syntax of the plotting system in the UNIX version was inherited from the DOS version. Although the UNIX version does not require all the information that the DOS version does, it was decided to keep the plotting system backwards compatible[4]. All fields in the plot structure should thus be filled in even though the UNIX version of Fuz only requires one of them.

The format of the plot structure is as follows:

```
plot Graph_Title
X-axis variable
min_x_value max_x_value
x_subdivisions
Y-axis variable
min_y_value max_y_value
y_subdivisions
endplot
```

An example of a plot structure is:

```
plot Y vs Tyd
tyd
-1 1
10
y
0 1
10
endplot
```

Fuz for UNIX only uses the x-axis variable when generating statistics for OPNET®. Please refer to ATM7.FUZ on the disk for a complete OPNET® example.

[4] The author did not feel that rewriting the plotting structure for the UNIX version would serve much purpose.

## 2. Integrating Fuz into OPNET®

The integration of Fuz into OPNET® includes:

1. How to insert Fuz into the simulation.
2. The format of the input file.
3. How to register OPNET® variables with Fuz.
4. Changing fuzzy sets on the fly.

### 2.1 Inserting Fuz into OPNET®

Appendix E presents the nodes and their corresponding states that make up the entire simulation. Figure 2-1 shows the layout of the one node that forms the states of the fuzzy logic controller.

Figure 2-1 shows that at start-up the controller is initialised and then goes into the idle state. It waits there until it is time for the next fuzzyfication process or the end of the simulation occurs. If it's the start of a new fuzzyfication process then the variable values are collected from the other nodes in the controller. The program then proceeds to execute the fuzzy logic engine. Here the variables are fuzzyfied, the rule base is applied and the output variable(s) is (are) defuzzyfied. The control action is then applied in the action state, after which the next fuzzyfication process is scheduled in the sched state.

Fuz is inserted into the controller by loading FUZ.INC (on the accompanying disk) into the functional block of this node in OPNET®.

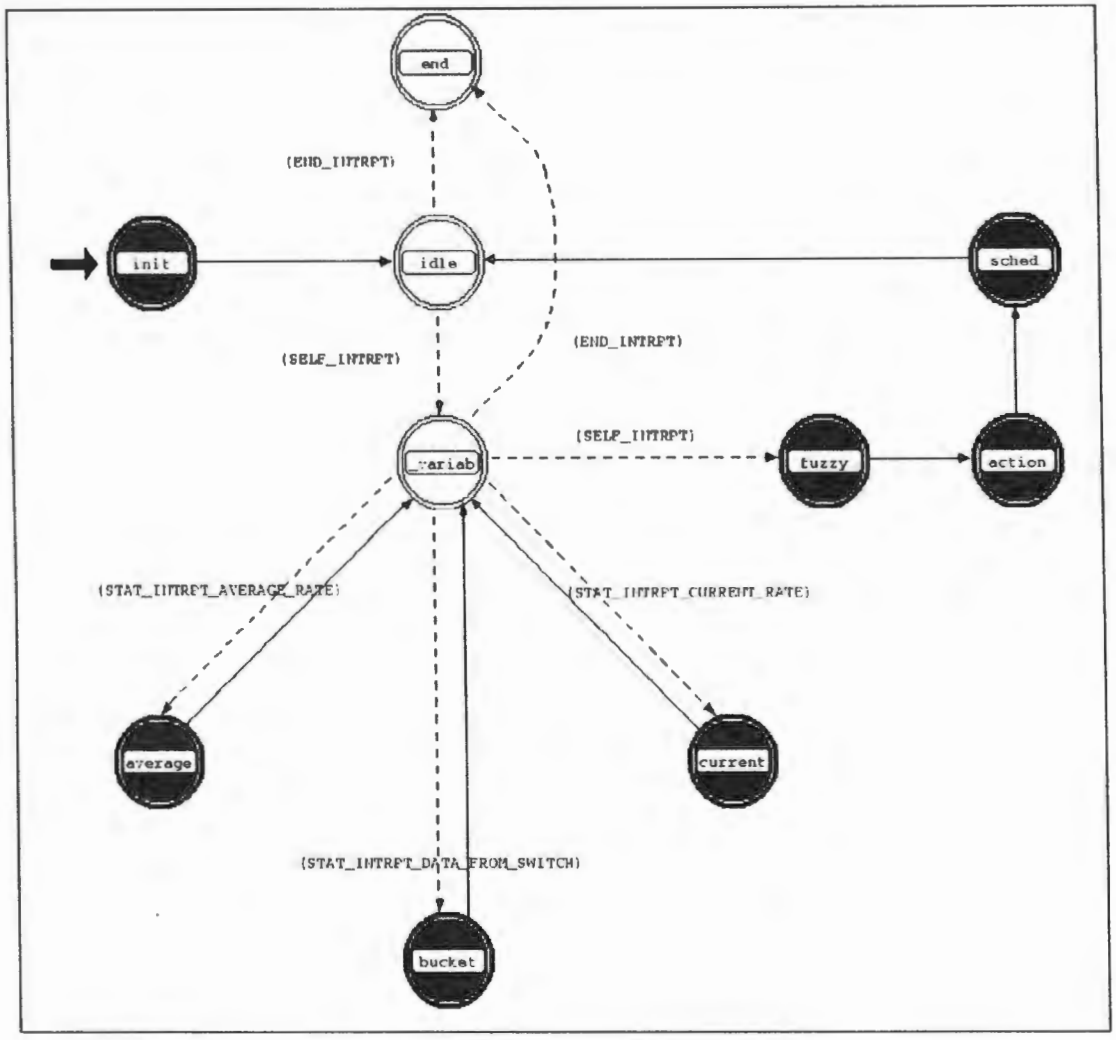



Figure 2-1 Fuzzy Controller states.

## 2.2 The Input File

The input file is a remnant of the interactive environment of the DOS version of Fuz, and as such contains the same commands as before.

The commands are:

| | |
|---|---|
| Load | • Deletes the previous fuzzy file (if any was loaded).<br>• Load a fuzzy file. The next line must contain the full name of the file to be loaded. |
| Rules | • Displays the names of rules that have been loaded. |

| | |
|---|---|
| Reset | • Resets Fuz.<br>• Initialises all input fuzzy variables to their start-up values. |
| Run | • Executes the fuzzy program. |
| Cons | • Displays all constants that are defined in the system. |
| Fvars | • Displays the names of fuzzy variables that have been loaded. |
| Exit | • Quits the fuzzy program. |

The input file is called FUZZY.INP and typically contains the following to load a fuzzy file called ATM7.FUZ:

```
load
atm7.fuz
reset
run
exit
```

### 2.3 Registering OPNET® variables with Fuz

Previously it was determined that Fuz needed to have access to state variables defined in OPNET®. This is done by providing Fuz with the address of the OPNET® state variable and a label by which it can be identified. The information must be provided in the *init* process of the OPNET® fuzzy controller (refer to Figure 2-1 and Appendix E).

A registration example would be:

```
/*Let the Fuzzy Logic Controller know about the OPNET variables it needs.*/
/*bucket_count and decision are registered as BUCKET_COUNT and
DECISION respectively in Fuz. */

op_register_state_variable("BUCKET_COUNT", &(bucket_count));
op_register_state_variable("DECISION", &(decision));
```

Registration typically takes place in the *init* state of Figure 2-1.

The registered variables can then be used in normal Fuz equations by preceding the variable's label with a backslash (that is a \), for example

```
eqn        x=\bucket_count+3,
```

and as the x-variable in plot structures, for example

```
plot OPNET's bucket_count variable.
\bucket_count
-1 1
10
y
0 1
10
endplot
```

### 2.4 Modifying Fuzzy Sets

It is also possible to change the shape and position of fuzzy sets after the Fuz file has been loaded. This is useful if the fuzzy sets need to be optimised or changed as a simulation is in progress. Only predefined fuzzy sets may be modified, that is this function can not be used to create new sets.

The op_change_fuzzy_set function in Fuz is used to modify a fuzzy set, for example

```
op_change_fuzzy_set( "BUCKET", "L",
                     0.0, 1.0,
                     1.0, 0.5,
                     1.5, 0.5,
                     2.0, 0.0);
```

takes an existing fuzzy set L of fuzzy variable BUCKET and sets its first, second, third and fourth co-ordinates to < 0.0 1.0 >, < 1.0 0.5 >, < 1.5 0.5 > and < 2.0 0.0 > respectively.

The range of the fuzzy variable can also be changed using the op_change_fuzzy_var_range function of Fuz, e.g.

```
op_change_fuzzy_var_range("SA", -3.0, 3.0);
```

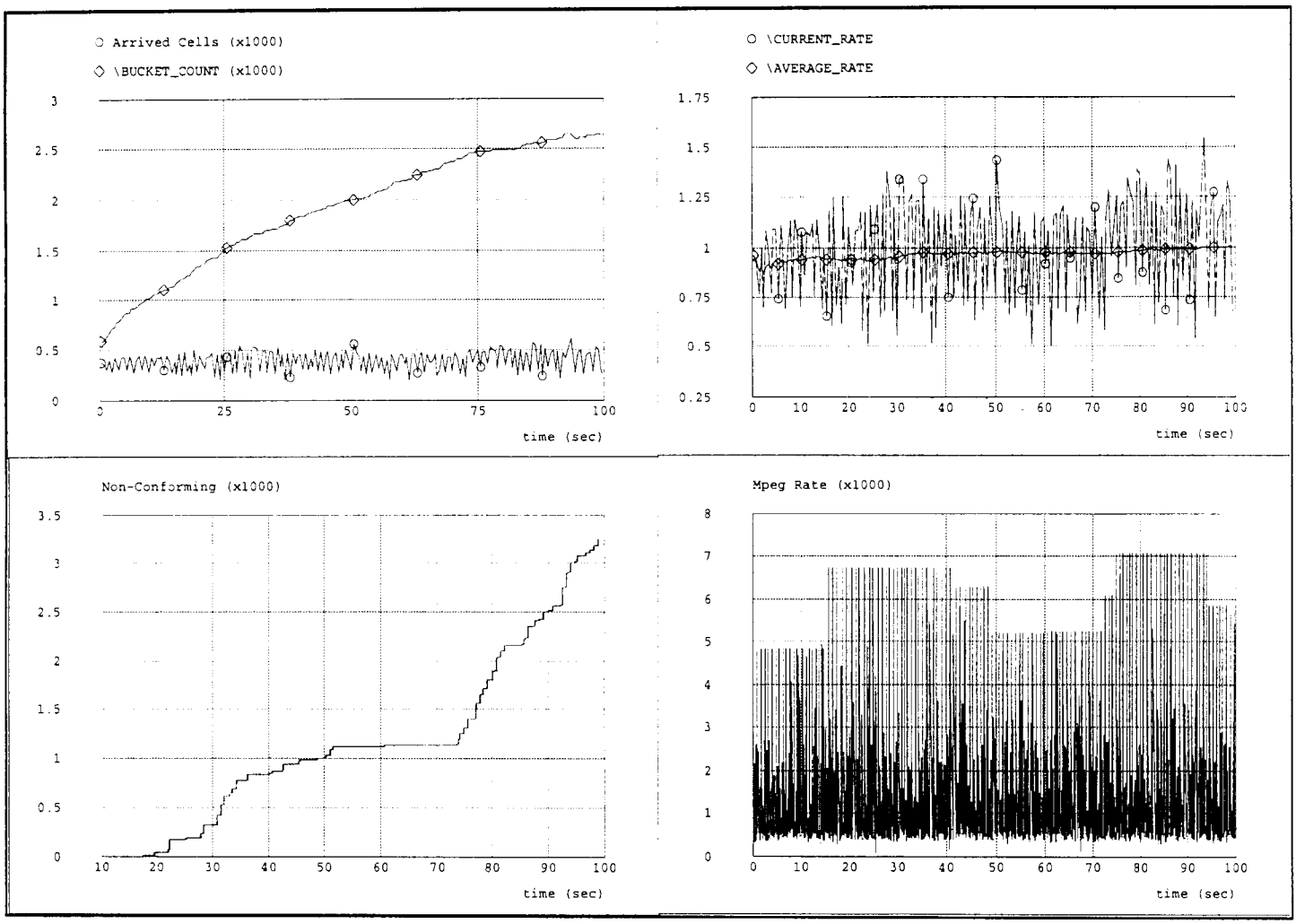
changes the range of the fuzzy variable SA to a minimum of -3.0 and a maximum of 3.0. An example of this can be found in the init process of Figure 2-1 (also refer to Appendix E).

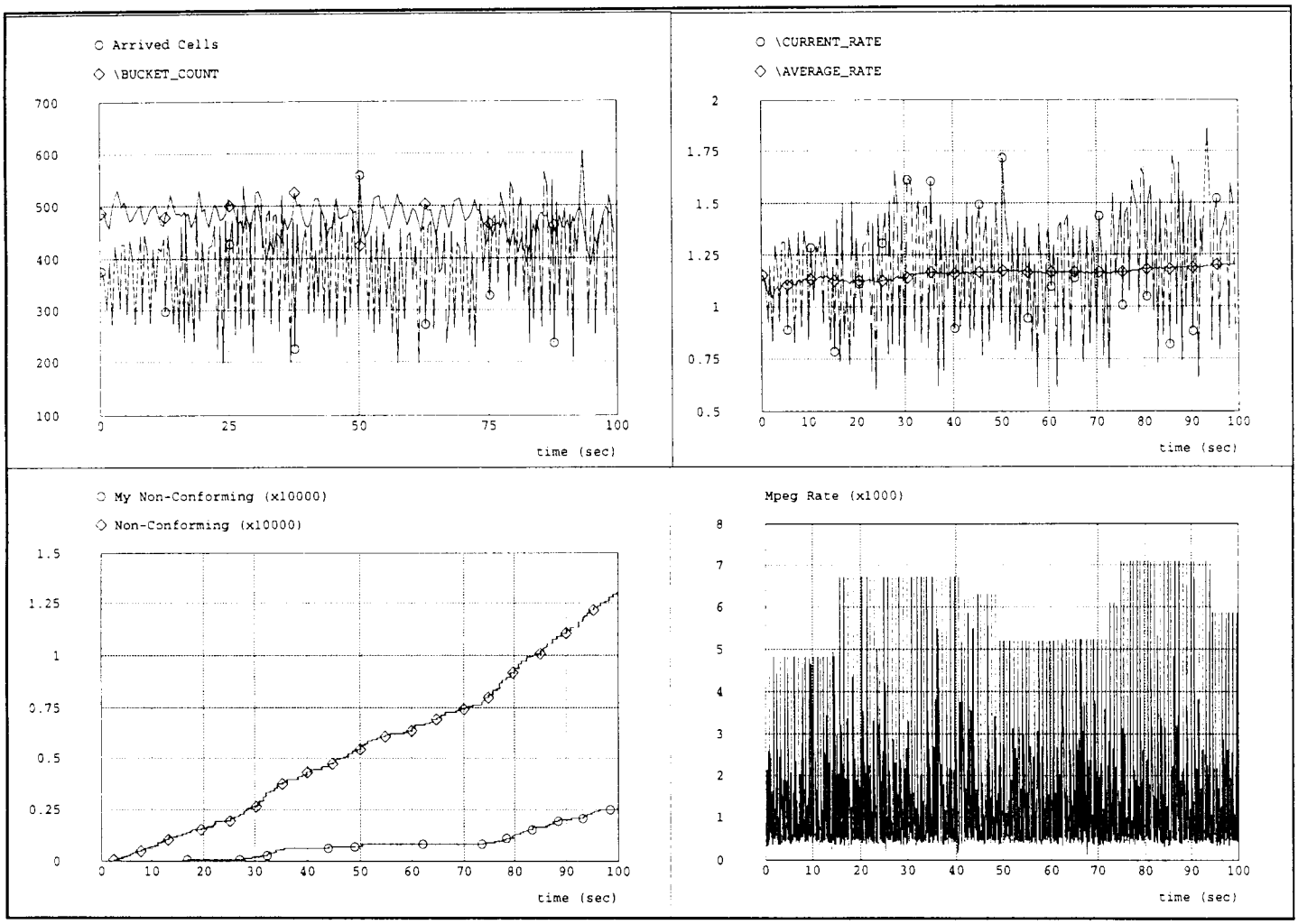
--oooOOOooo---

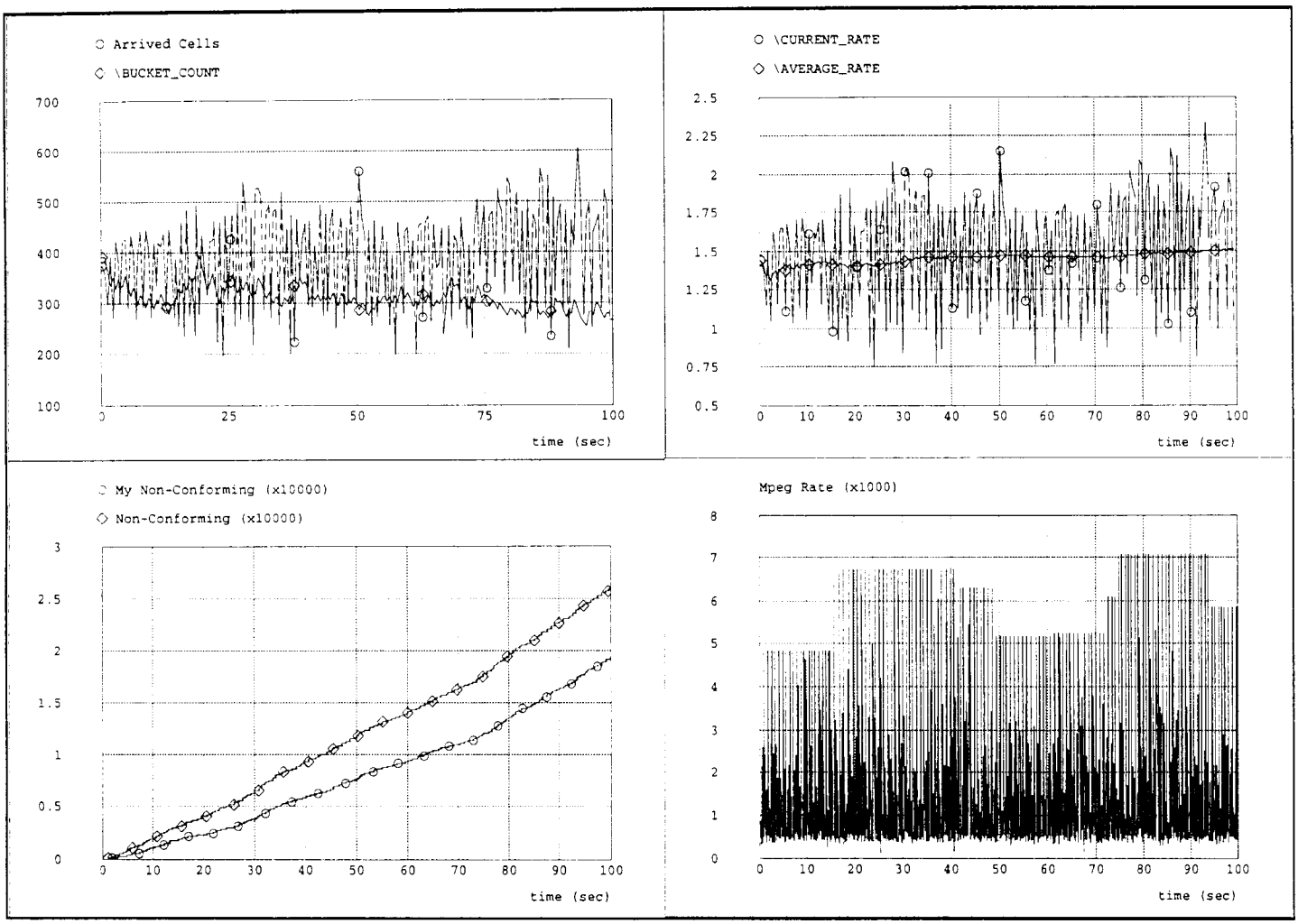
# APPENDIX C

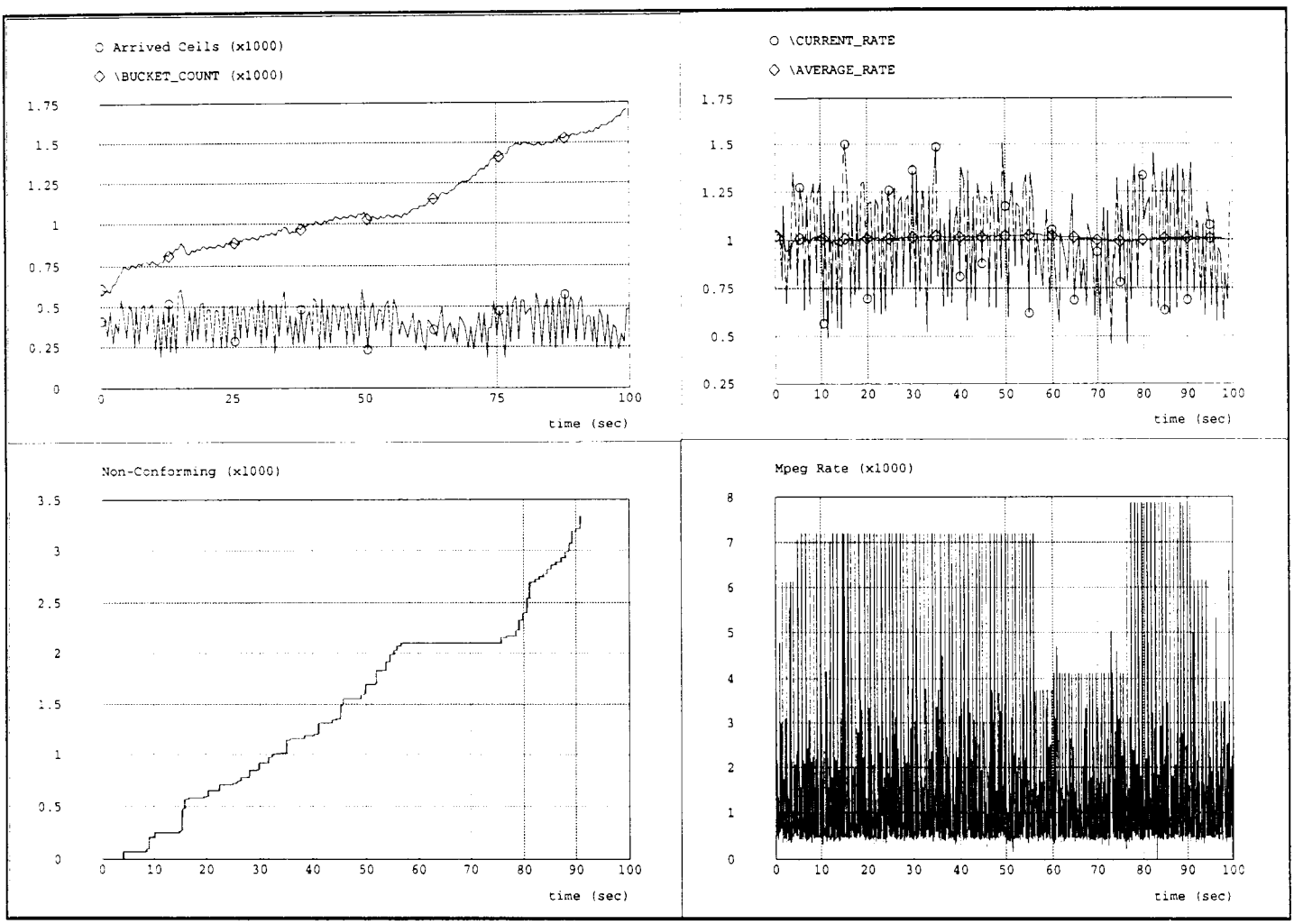
## DETAILED SIMULATION RESULTS

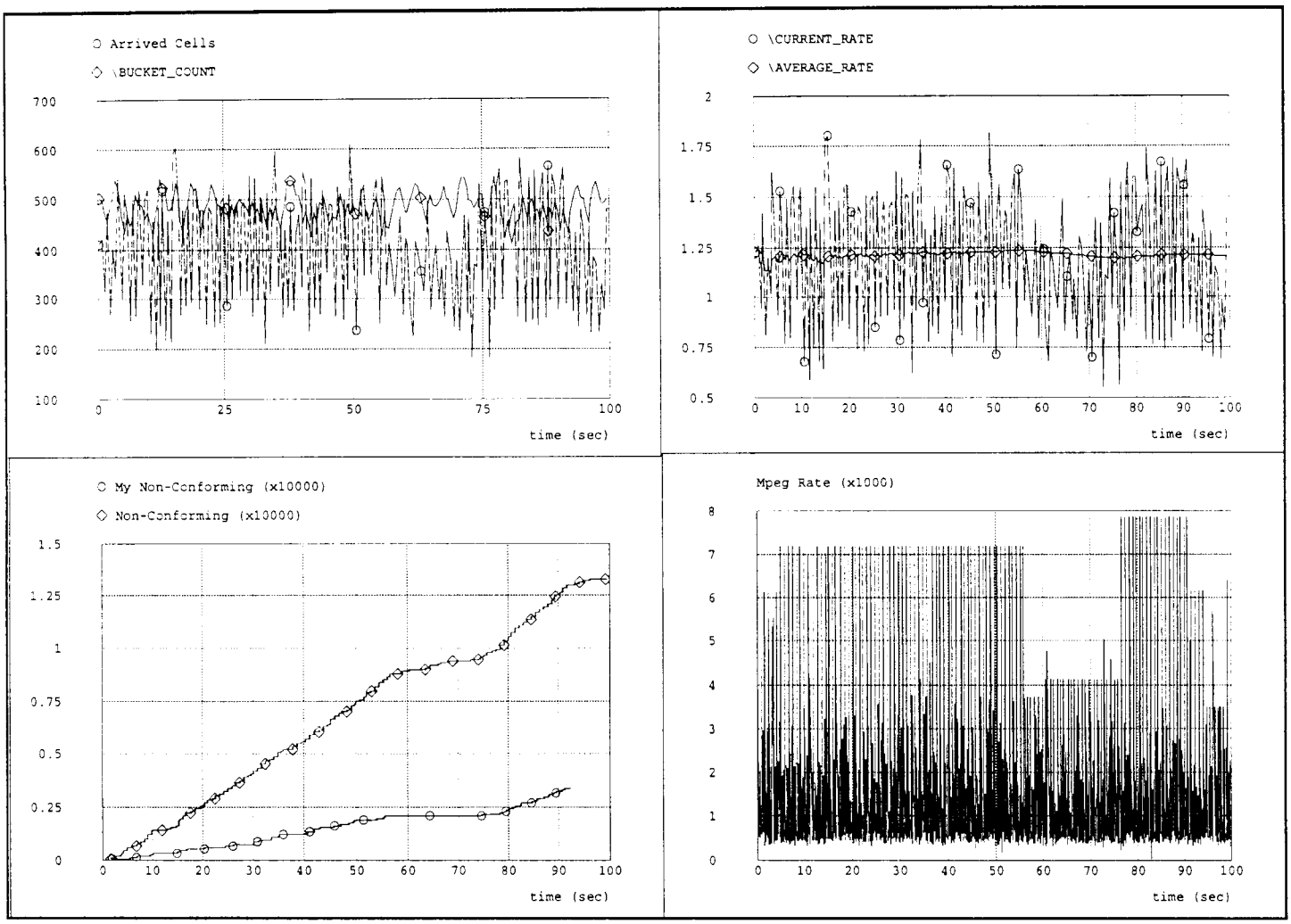
| Test | m | Seed | NPCR | Number of cells | AGR | NAGR | IBS (fuzzy) | BS (GCRA) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 100 | 7080 | 78113 | 781.1 | 781.1 | 585.8 | 236 |
| 2 | 1.2 | 100 | 7080 | 78113 | 781.1 | 650.9 | 488 | 236 |
| 3 | 1.5 | 100 | 7080 | 78113 | 781.1 | 520.7 | 390.5 | 236 |
| 4 | 1 | 200 | 7860 | 80239 | 802.4 | 802.4 | 601.8 | 262 |
| 5 | 1.2 | 200 | 7860 | 80239 | 802.4 | 668.7 | 501.5 | 262 |
| 6 | 1.5 | 200 | 7860 | 80239 | 802.4 | 534.9 | 401.2 | 262 |
| 7 | 1 | 300 | 8250 | 75363 | 753.6 | 753.6 | 565.2 | 275 |
| 8 | 1.2 | 300 | 8250 | 75363 | 753.6 | 628 | 471 | 275 |
| 9 | 1.5 | 300 | 8250 | 75363 | 753.6 | 502 | 376 | 275 |
| 10 | 1 | 400 | 6840 | 75014 | 750.1 | 750.1 | 562.6 | 228 |
| 11 | 1.2 | 400 | 6840 | 75014 | 750.1 | 625.1 | 468.8 | 228 |
| 12 | 1.5 | 400 | 6840 | 75014 | 750.1 | 500.1 | 375.1 | 228 |
| 13 | 1 | 500 | 6600 | 76986 | 769.9 | 769.9 | 577.4 | 220 |
| 14 | 1.2 | 500 | 6600 | 76986 | 769.9 | 641.6 | 481.2 | 220 |
| 15 | 1.5 | 500 | 6600 | 76986 | 769.9 | 513.3 | 385 | 220 |
| 16 | 1 | 600 | 7590 | 80174 | 801.7 | 801.7 | 601.3 | 253 |
| 17 | 1.2 | 600 | 7590 | 80174 | 801.7 | 668.1 | 501.1 | 253 |
| 18 | 1.5 | 600 | 7590 | 80174 | 801.7 | 534.5 | 400.9 | 253 |
| 19 | 1 | 700 | 7440 | 80012 | 800.1 | 800.1 | 600.1 | 248 |
| 20 | 1.2 | 700 | 7440 | 80012 | 800.1 | 666.8 | 500.1 | 248 |
| 21 | 1.5 | 700 | 7440 | 80012 | 800.1 | 533.4 | 400.1 | 248 |
| 22 | 1 | 800 | 7500 | 79137 | 791.4 | 791.4 | 593.6 | 250 |
| 23 | 1.2 | 800 | 7500 | 79137 | 791.4 | 659.5 | 494.6 | 250 |
| 24 | 1.5 | 800 | 7500 | 79137 | 791.4 | 527.6 | 395.7 | 250 |
| 25 | 1 | 900 | 7710 | 77912 | 779.1 | 779.1 | 584.3 | 257 |
| 26 | 1.2 | 900 | 7710 | 77912 | 779.1 | 649.3 | 487.0 | 257 |
| 27 | 1.5 | 900 | 7710 | 77912 | 779.1 | 519.4 | 389.6 | 257 |
| 28 | 1 | 1000 | 8760 | 82237 | 822.4 | 822.4 | 616.8 | 292 |
| 29 | 1.2 | 1000 | 8760 | 82237 | 822.4 | 685.3 | 514 | 292 |
| 30 | 1.5 | 1000 | 8760 | 82237 | 822.4 | 548.3 | 411.2 | 292 |

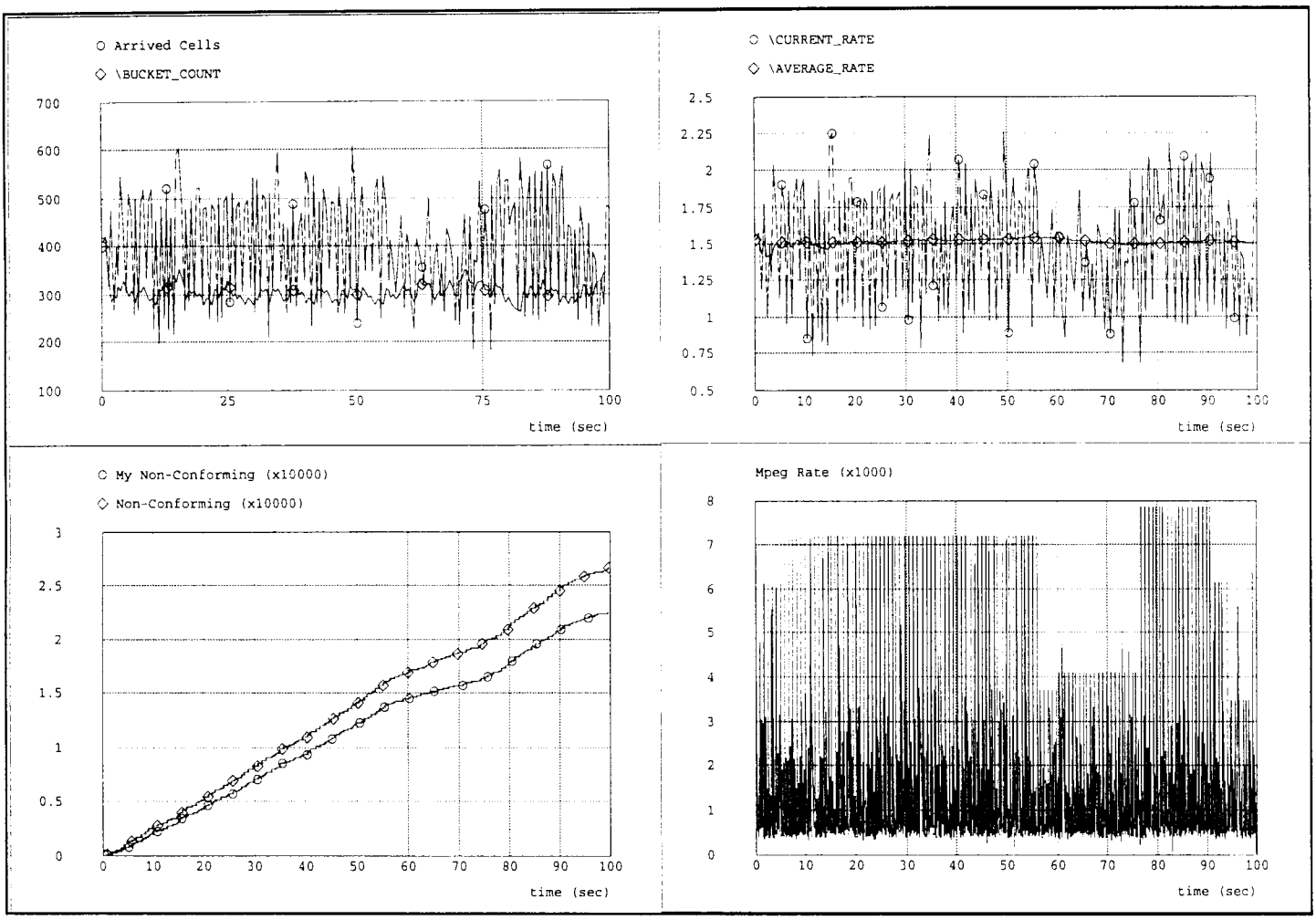

○ Arrived Cells (x1000)
◇ \BUCKET_COUNT (x1000)
3
2.5
2
1.5
1
0.5
0
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
1.75
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Non-Conforming (x1000)
3.5
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

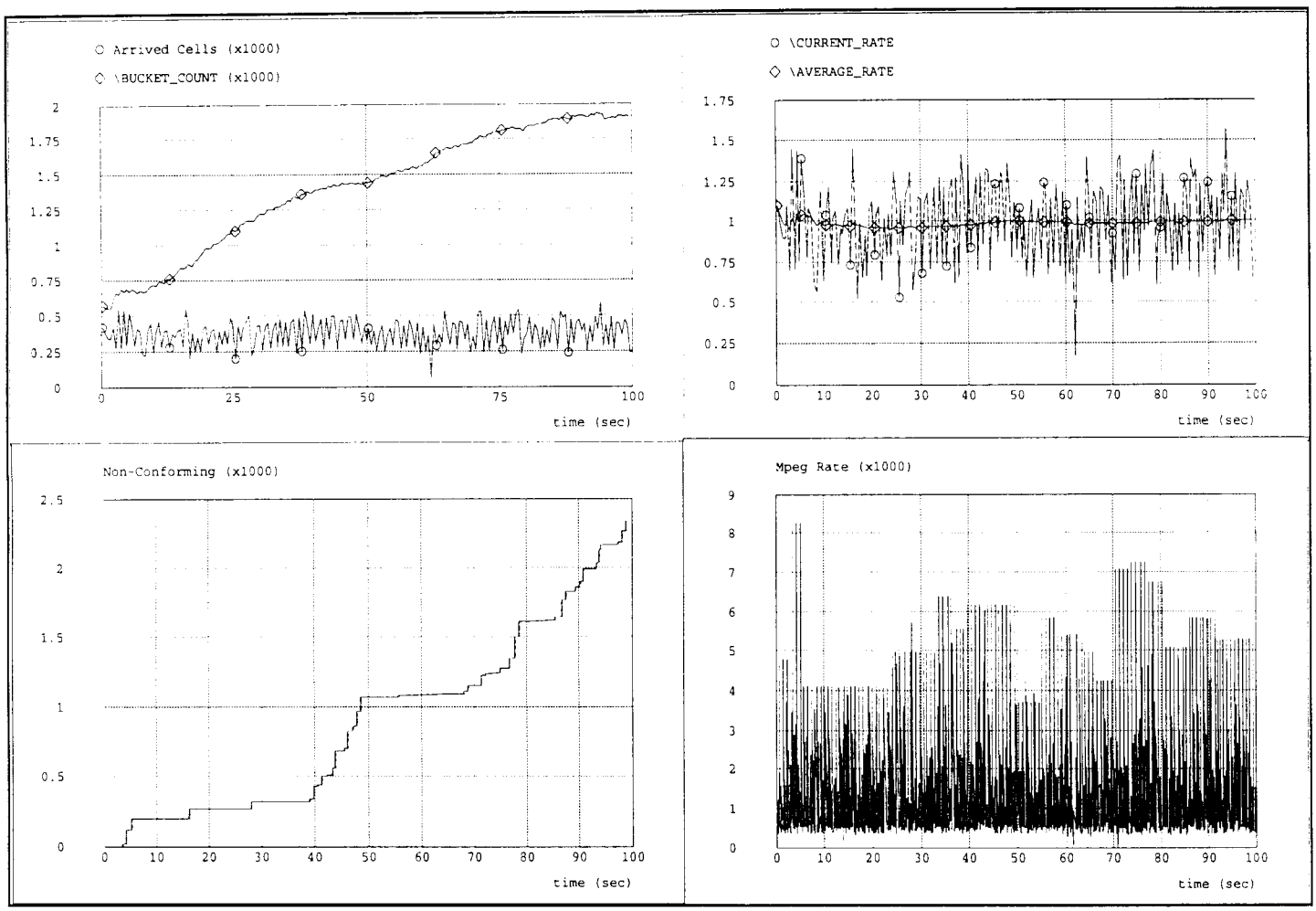

○ Arrived Cells
◇ \BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
○ My Non-Conforming (x10000)
◇ Non-Conforming (x10000)
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)

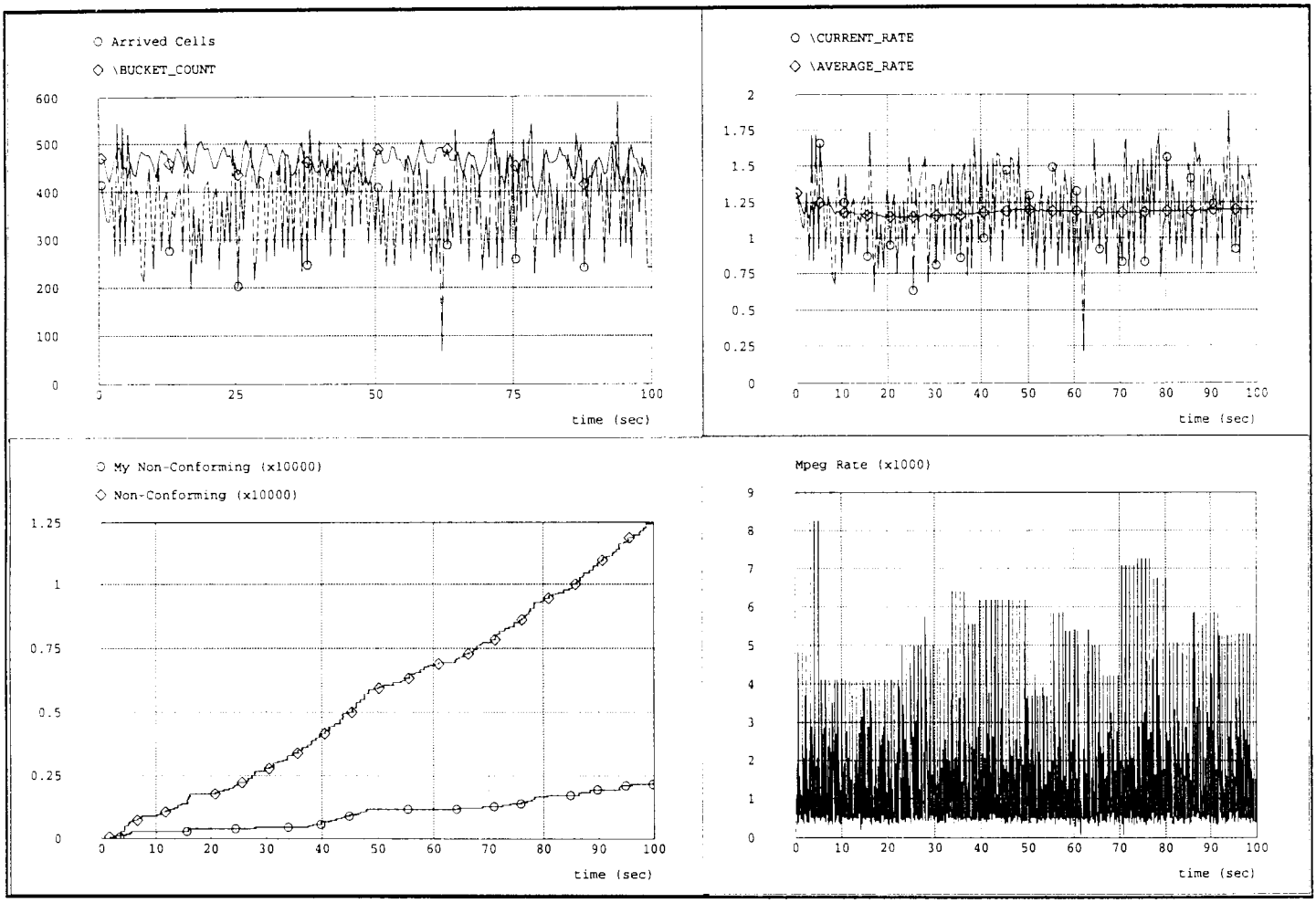

Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2.5
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)

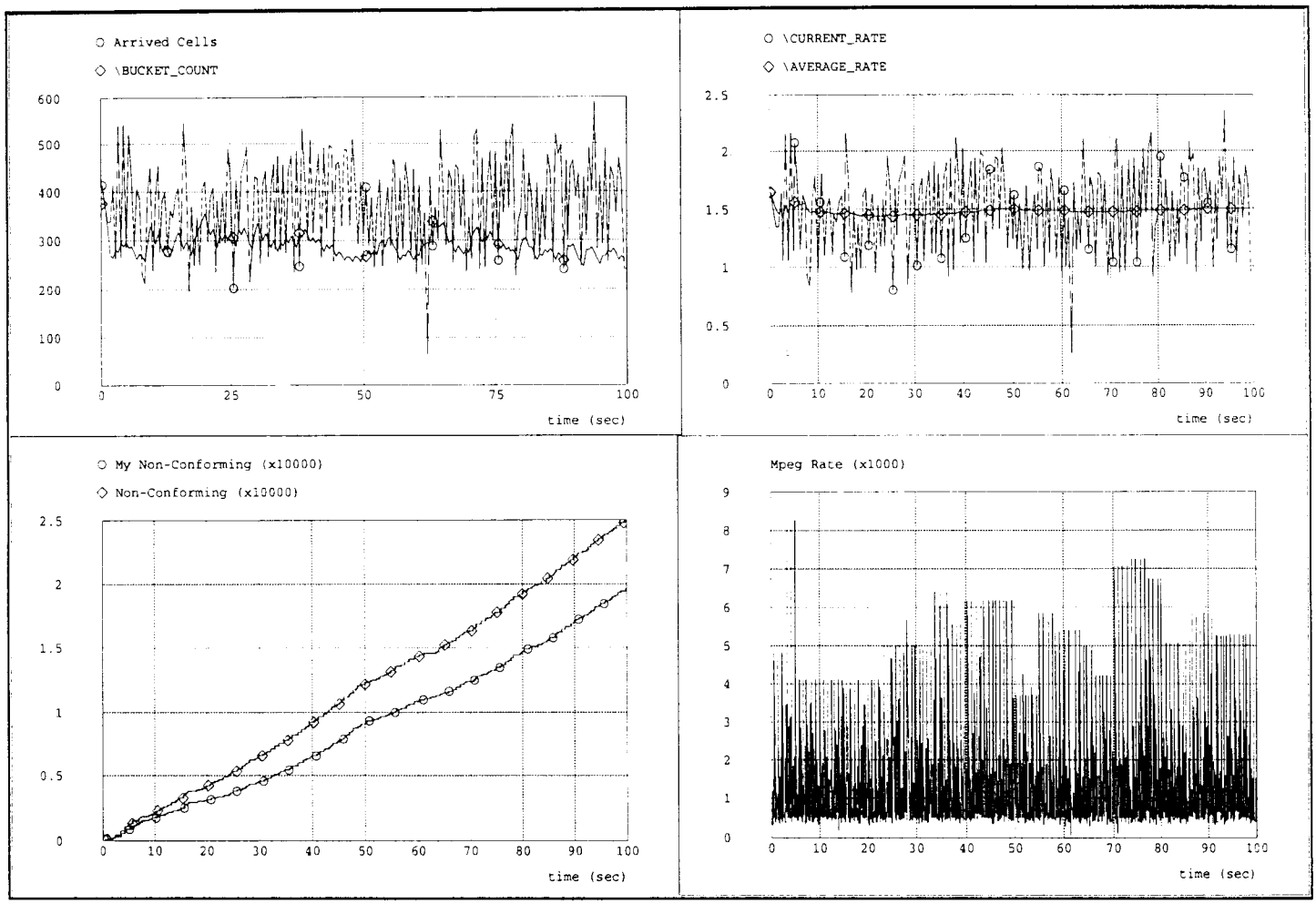

○ Arrived Cells (x1000)
◇ \BUCKET_COUNT (x1000)
1.75
1.5
1.25
1
0.75
0.5
0.25
0
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
1.75
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Non-Conforming (x1000)
3.5
3
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

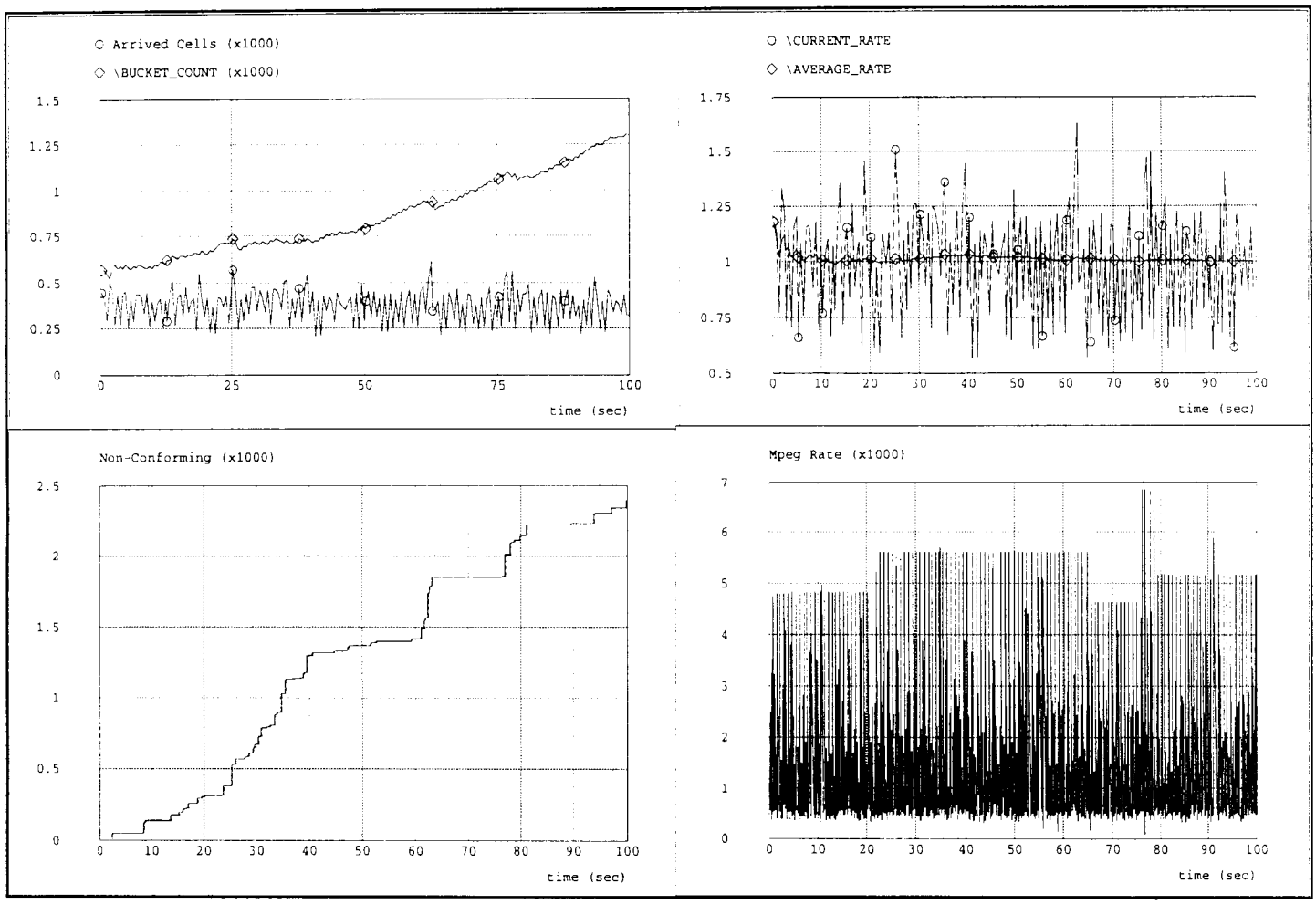

Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)

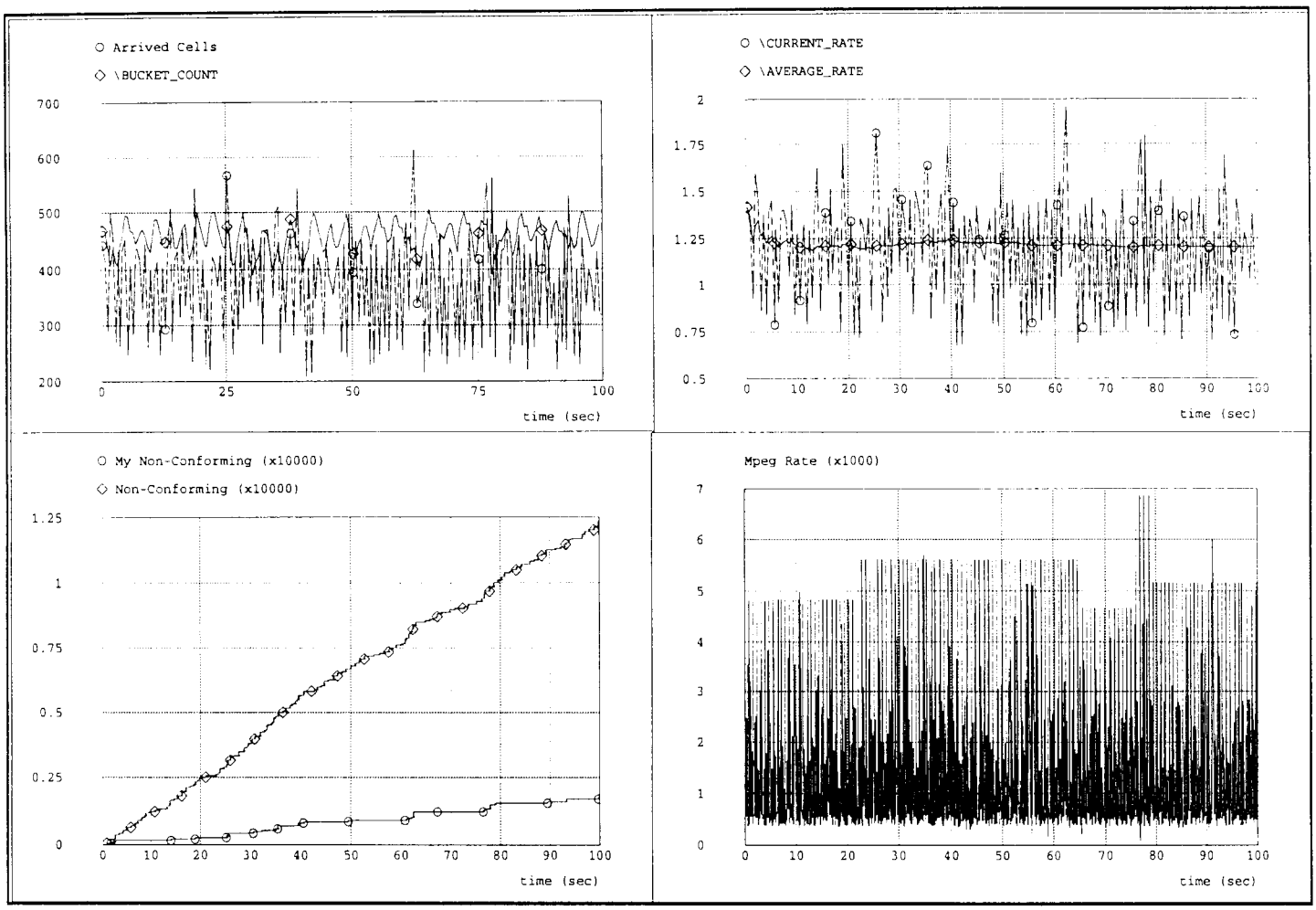

○ Arrived Cells
◇ \BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
2.5
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
○ My Non-Conforming (x10000)
◇ Non-Conforming (x10000)
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)

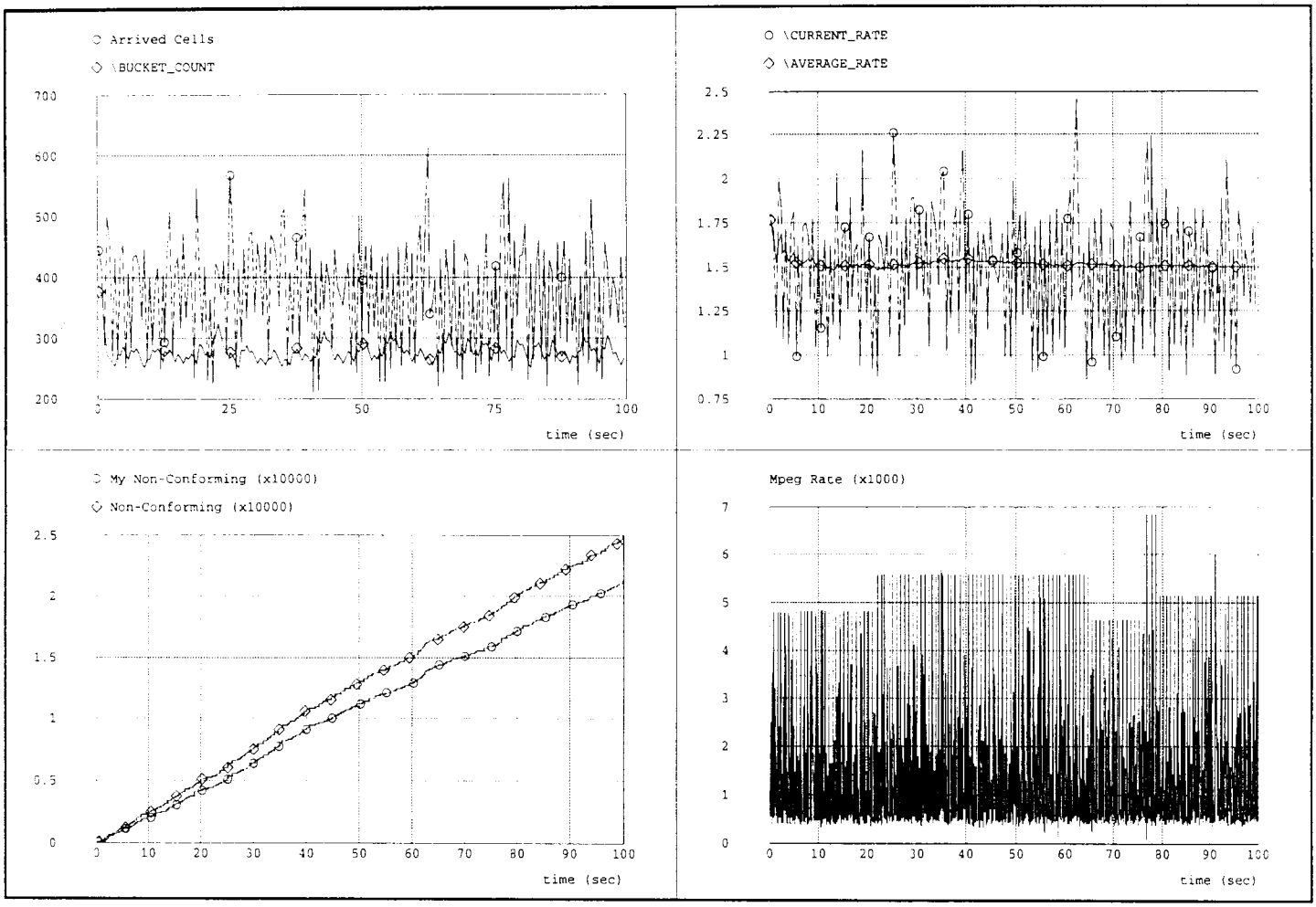

○ Arrived Cells (x1000)
◇ \BUCKET_COUNT (x1000)
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
1.75
1.5
1.25
1
0.75
0.5
0.25
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Non-Conforming (x1000)
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
9
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

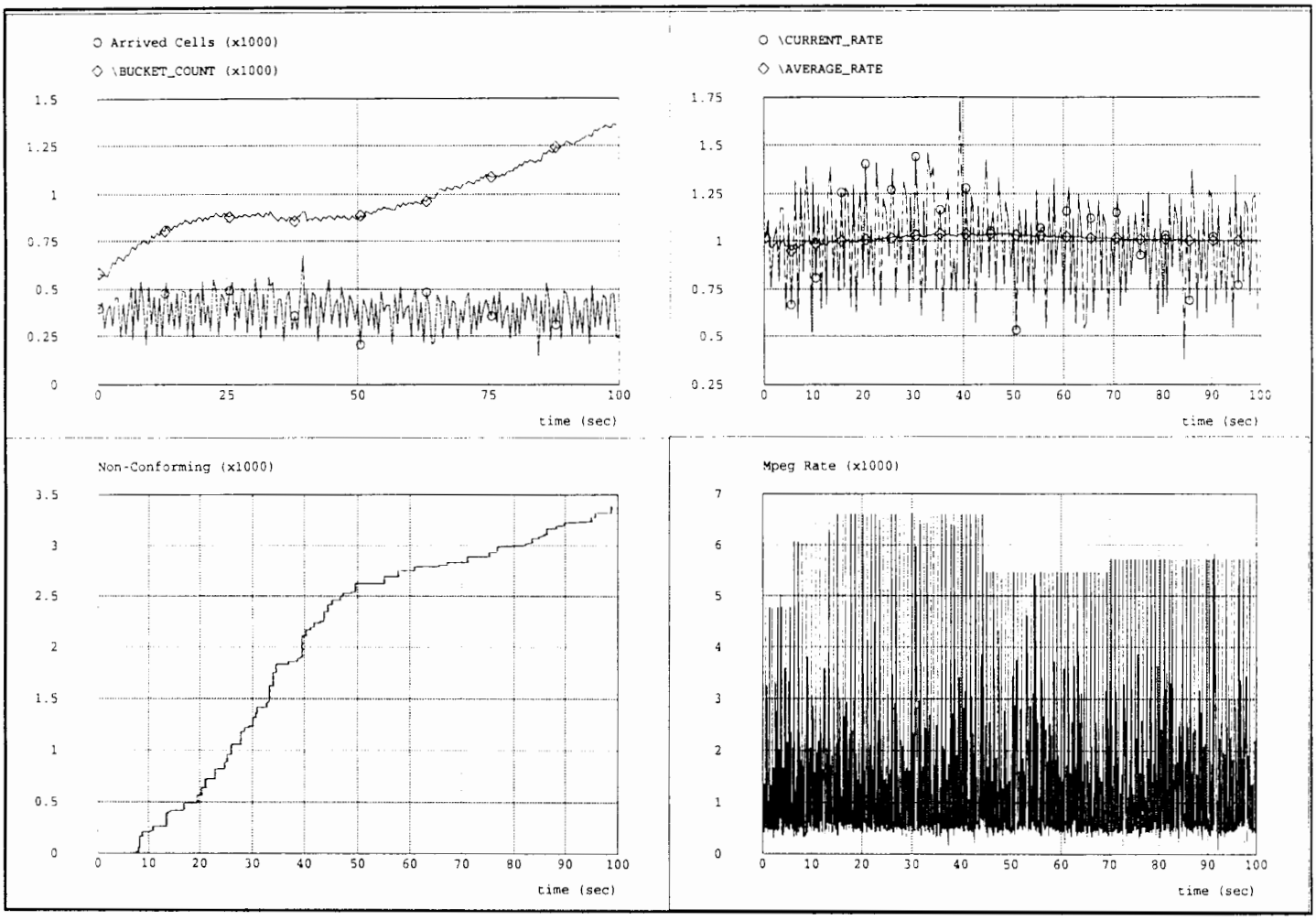

Arrived Cells
\BUCKET_COUNT
600
500
400
300
200
100
0
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
1.25
1
0.75
0.5
0.25
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
9
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

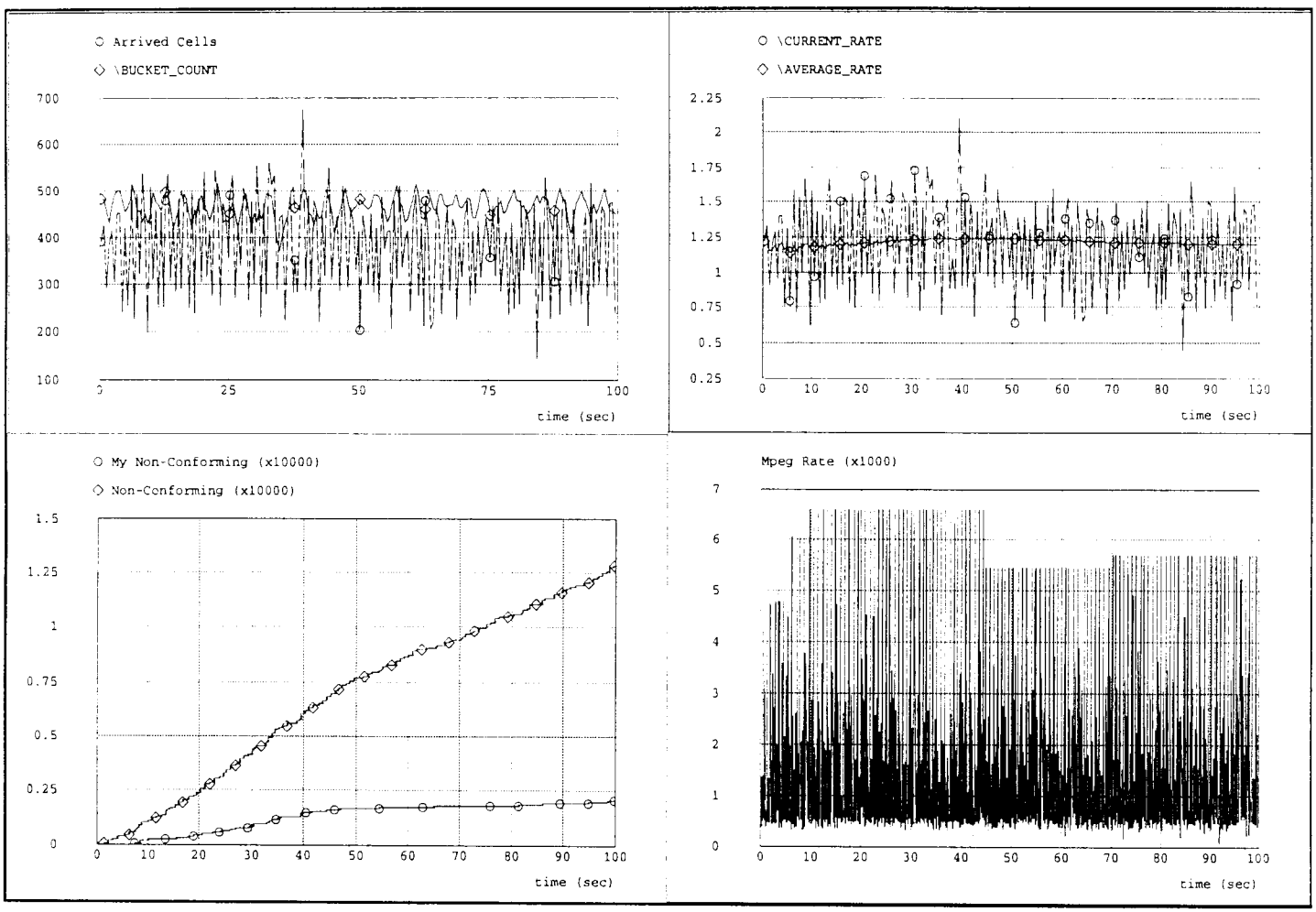

○ Arrived Cells
◇ \BUCKET_COUNT
600
500
400
300
200
100
0
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
○ My Non-Conforming (x10000)
◇ Non-Conforming (x10000)
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
9
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

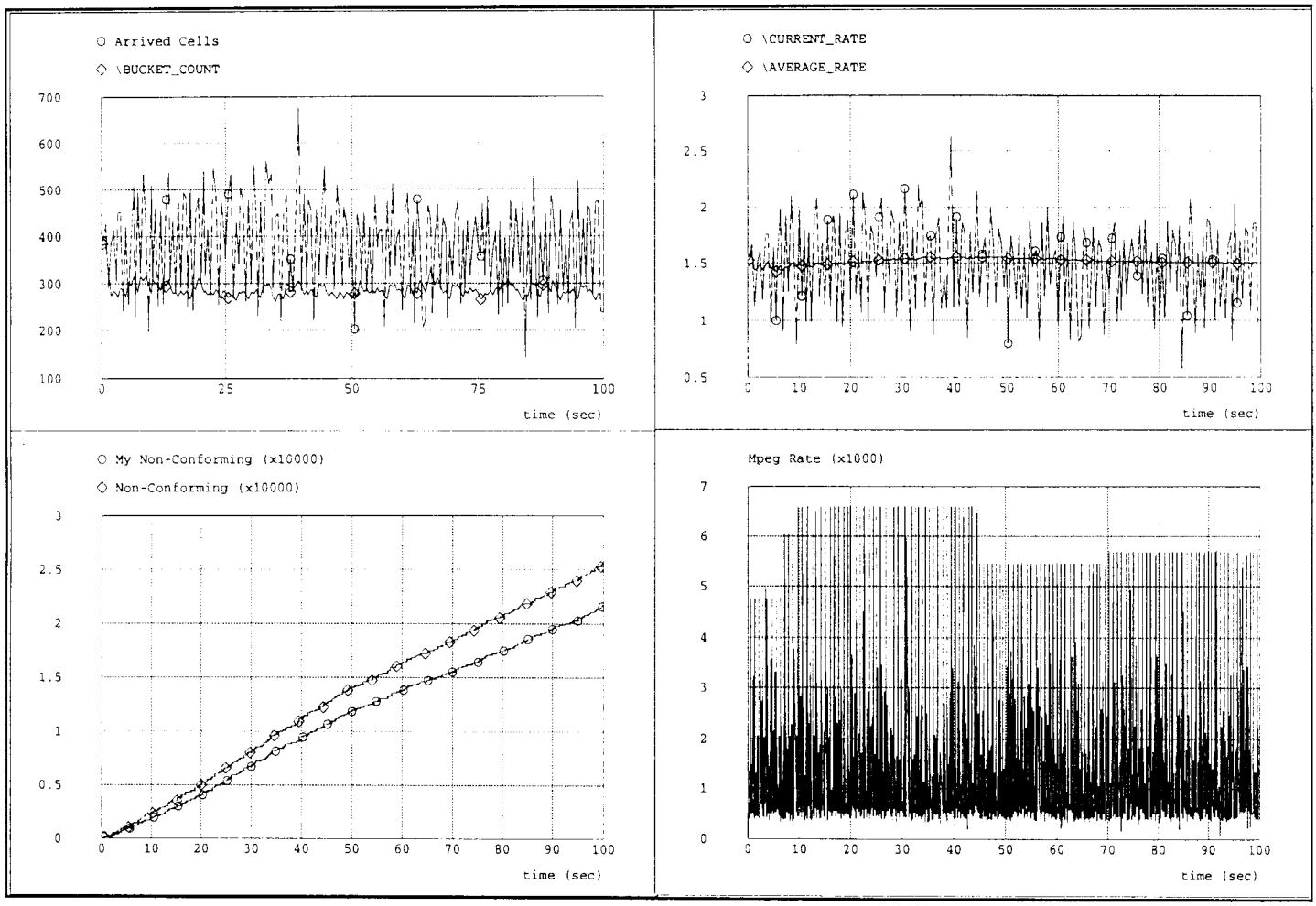

○ Arrived Cells (x1000)
◇ \BUCKET_COUNT (x1000)
1.5
1.25
1
0.75
0.5
0.25
0
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Non-Conforming (x1000)
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

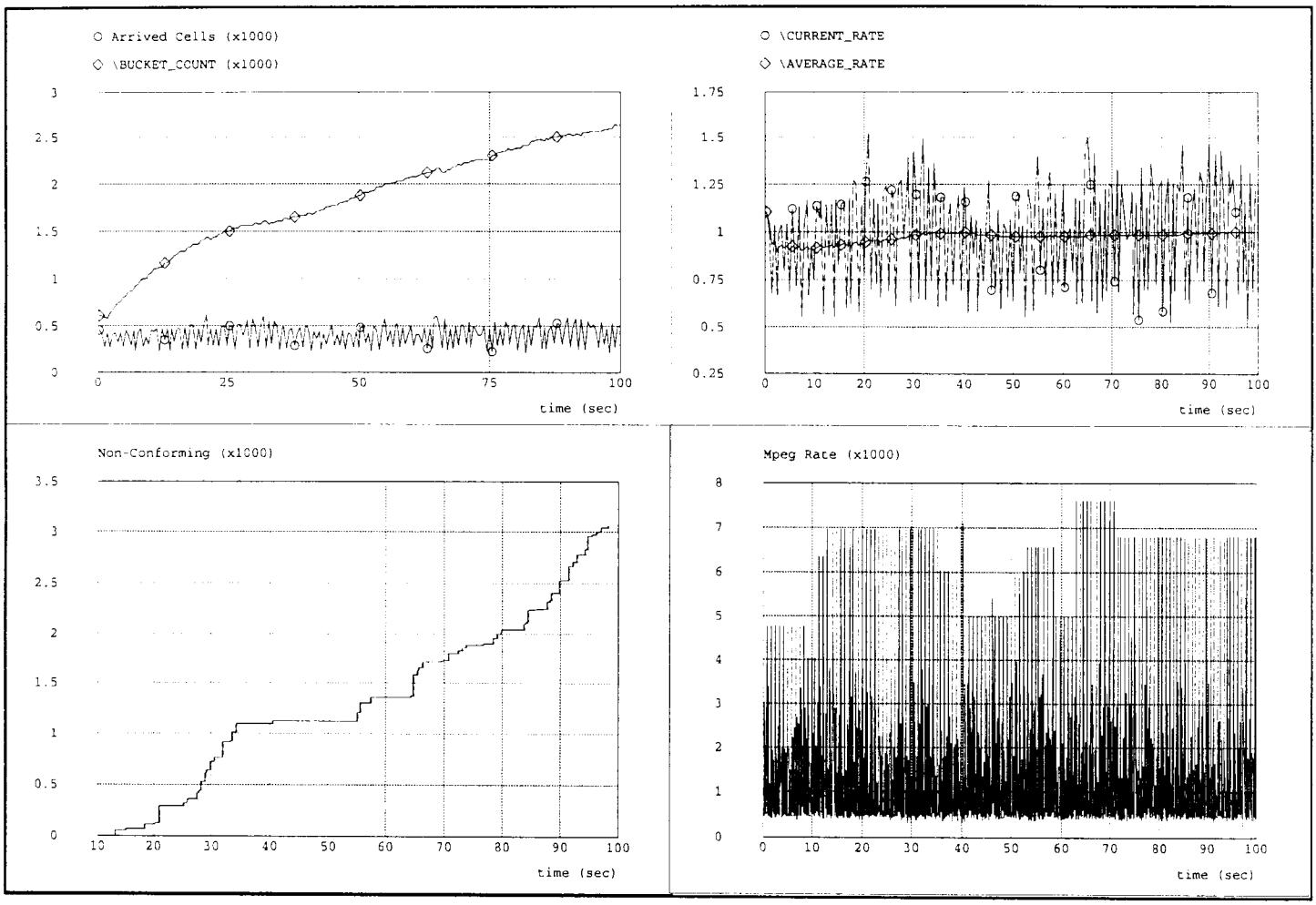

○ Arrived Cells
◇ \BUCKET_COUNT
700
600
500
400
300
200
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
○ My Non-Conforming (x10000)
◇ Non-Conforming (x10000)
1.25
1
0.75
0.5
0.25
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

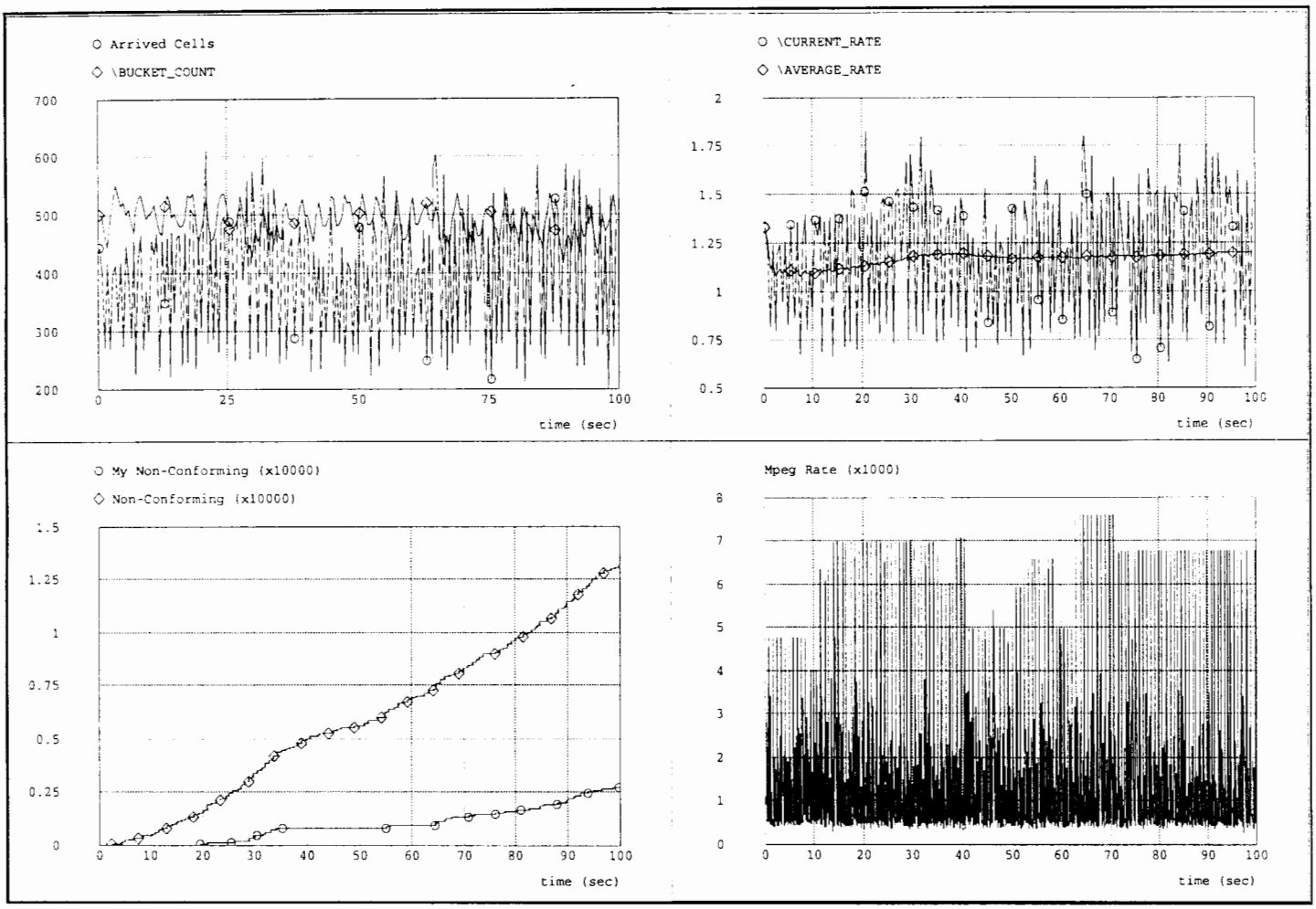

Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2.5
2.25
2
1.75
1.5
1.25
1
0.75
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

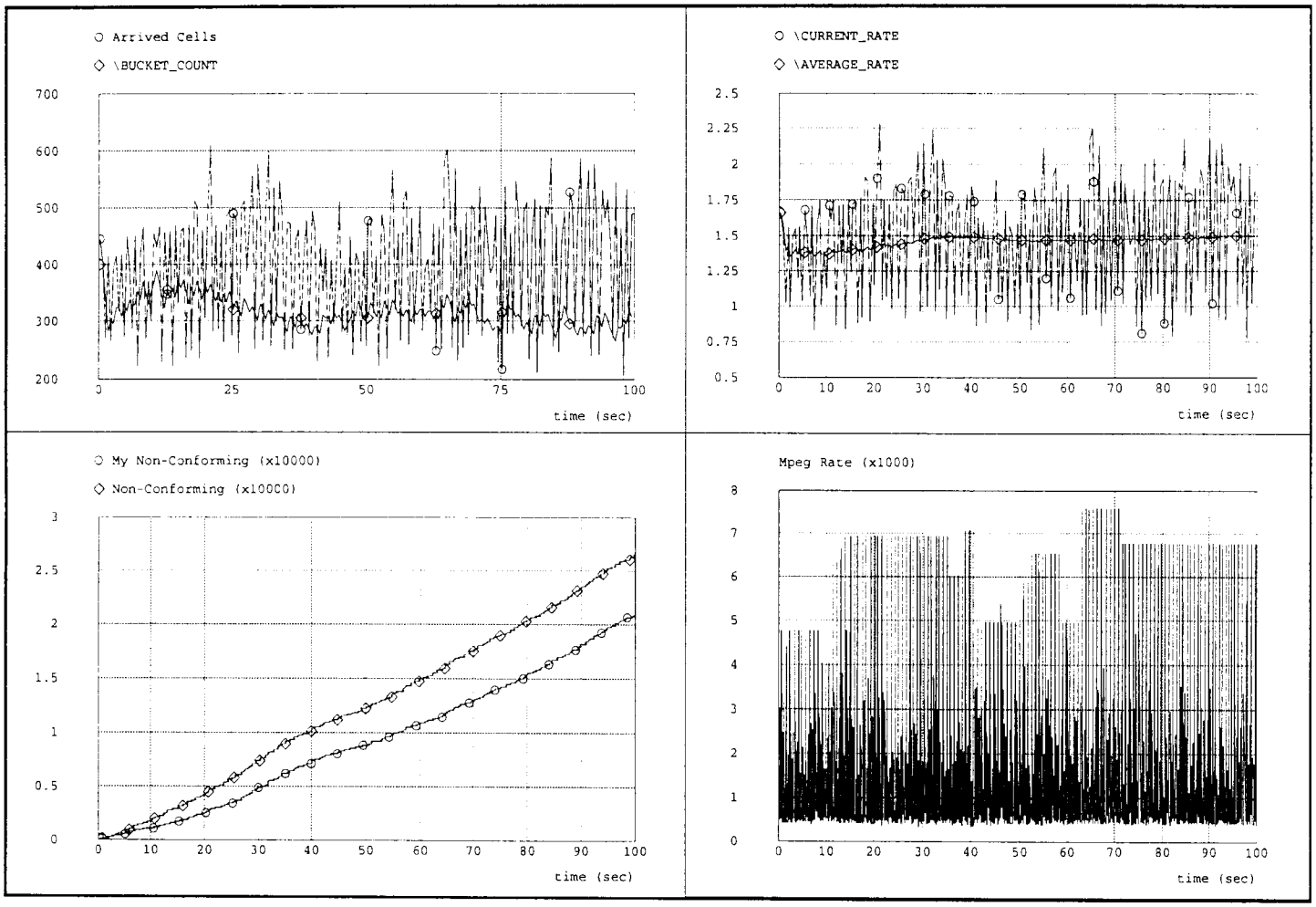

Arrived Cells (x1000)
\BUCKET_COUNT (x1000)
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
time (sec)
Non-Conforming (x1000)
time (sec)
Mpeg Rate (x1000)
time (sec)

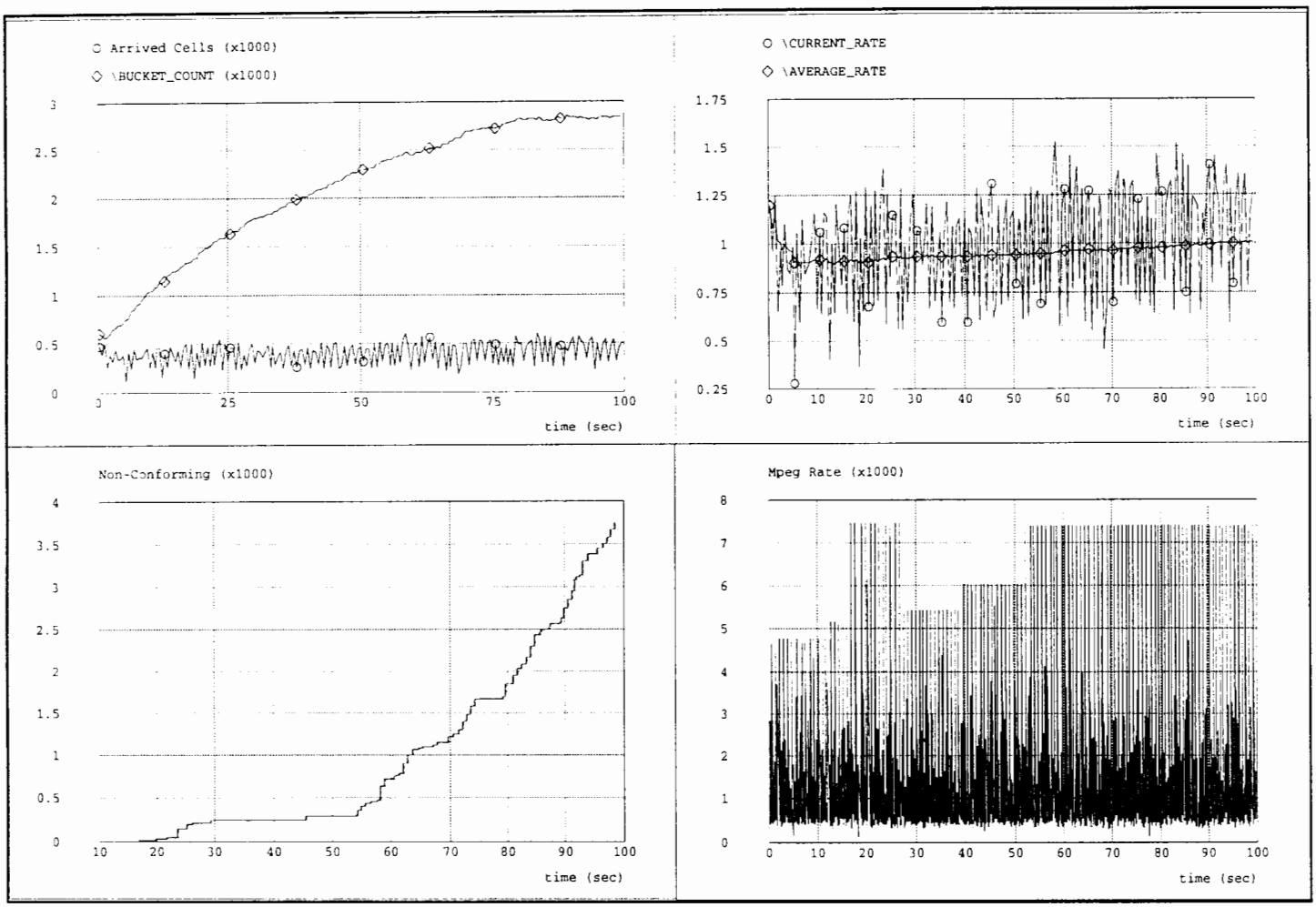

Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)

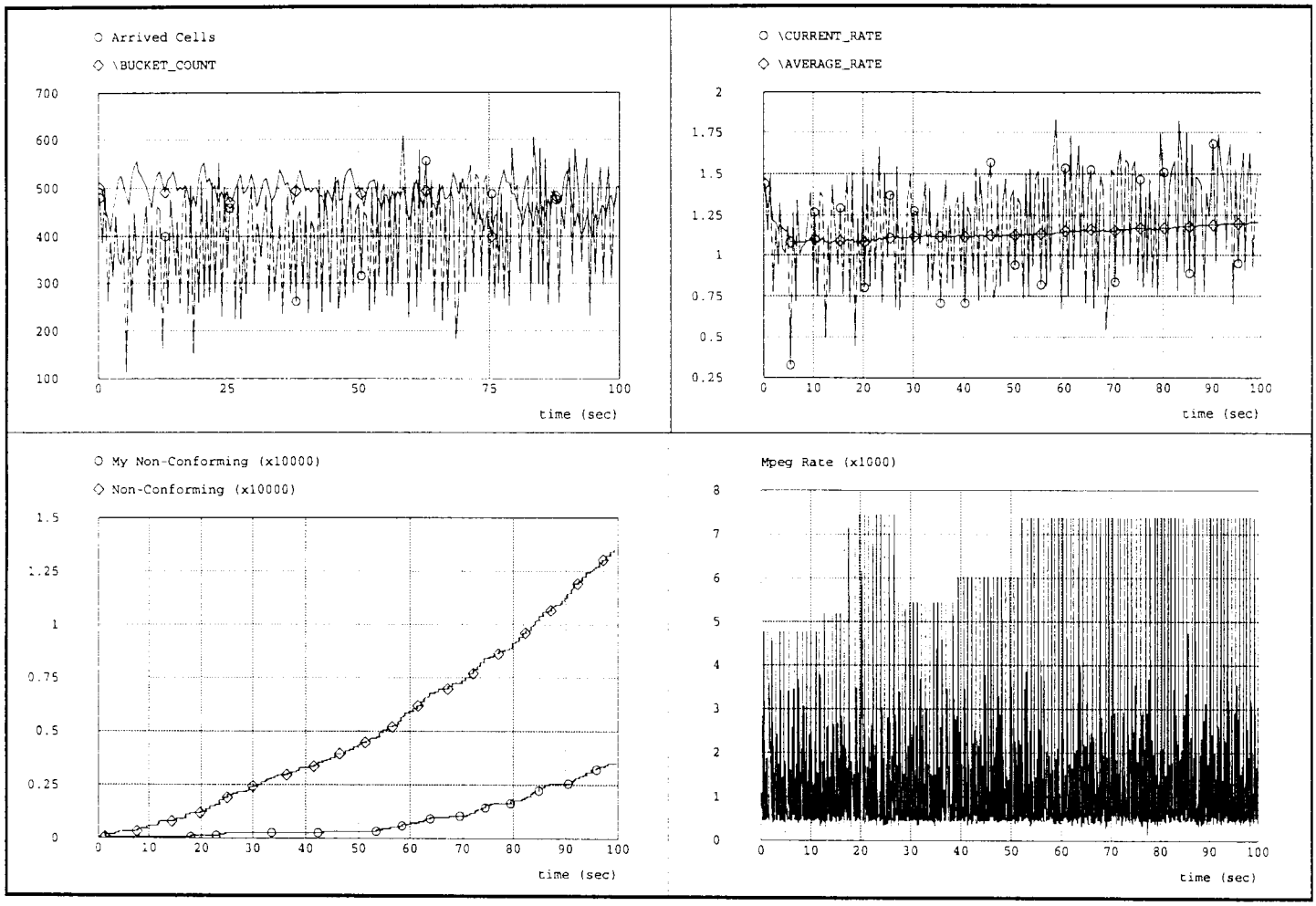

○ Arrived Cells
◇ \BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
○ My Non-Conforming (x10000)
◇ Non-Conforming (x10000)
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)

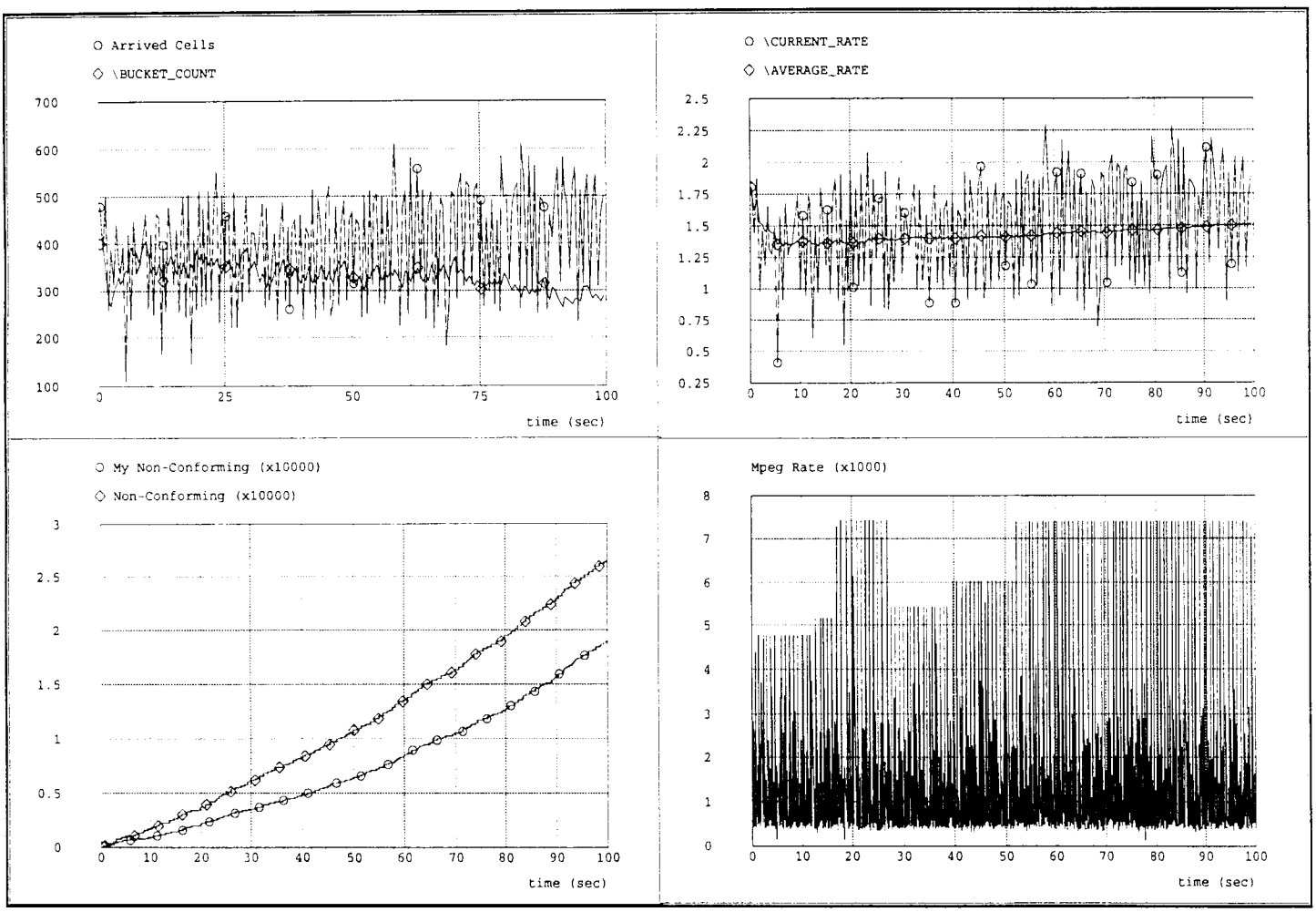

○ Arrived Cells (x1000)
◇ \BUCKET_COUNT (x1000)
3
2.5
2
1.5
1
0.5
0
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
1.75
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Non-Conforming (x1000)
3.5
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

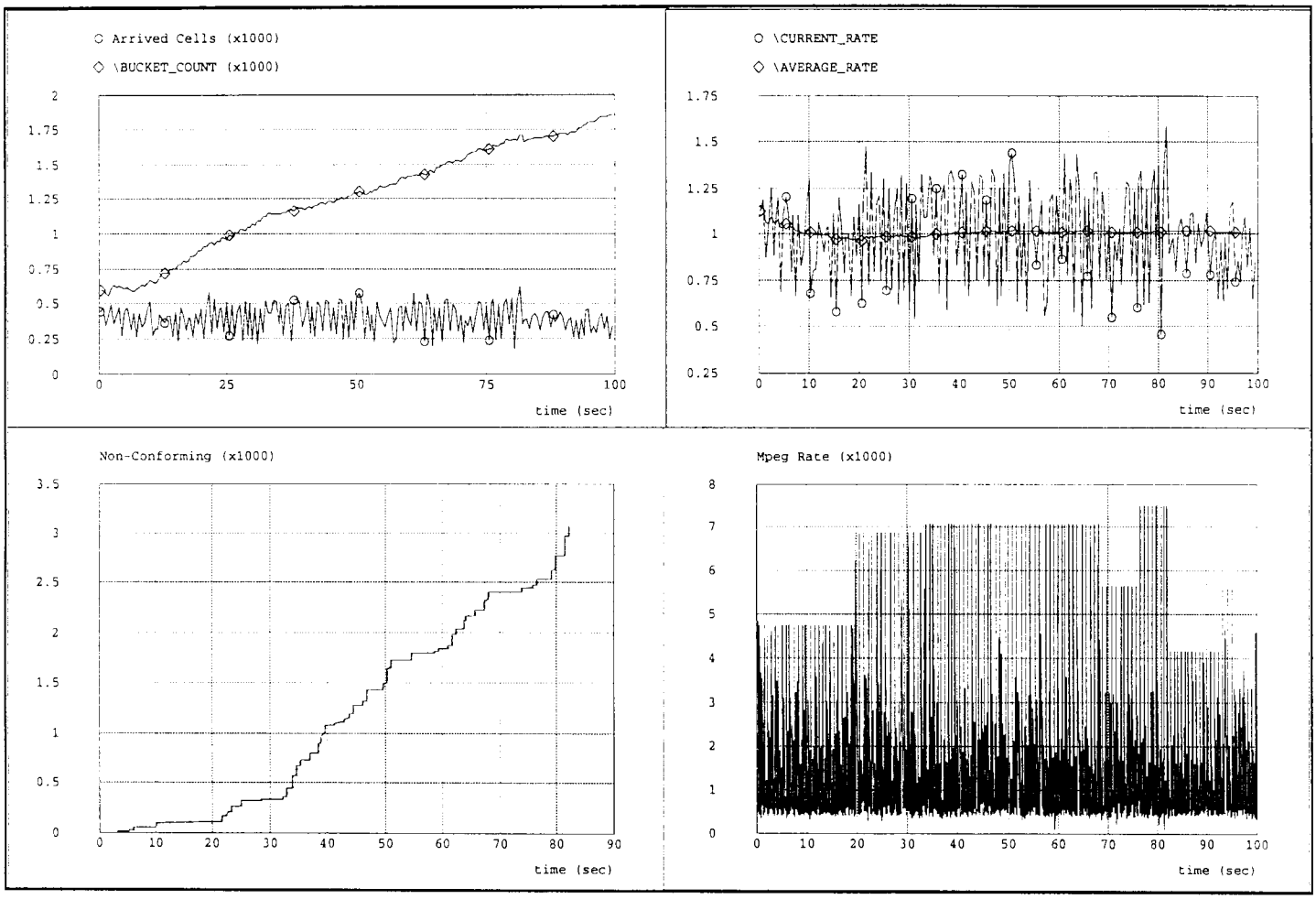

○ Arrived Cells
◇ \BUCKET_COUNT
700
600
500
400
300
200
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
○ My Non-Conforming (x10000)
◇ Non-Conforming (x10000)
1.5
1.25
1
0.75
0.5
0.25
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

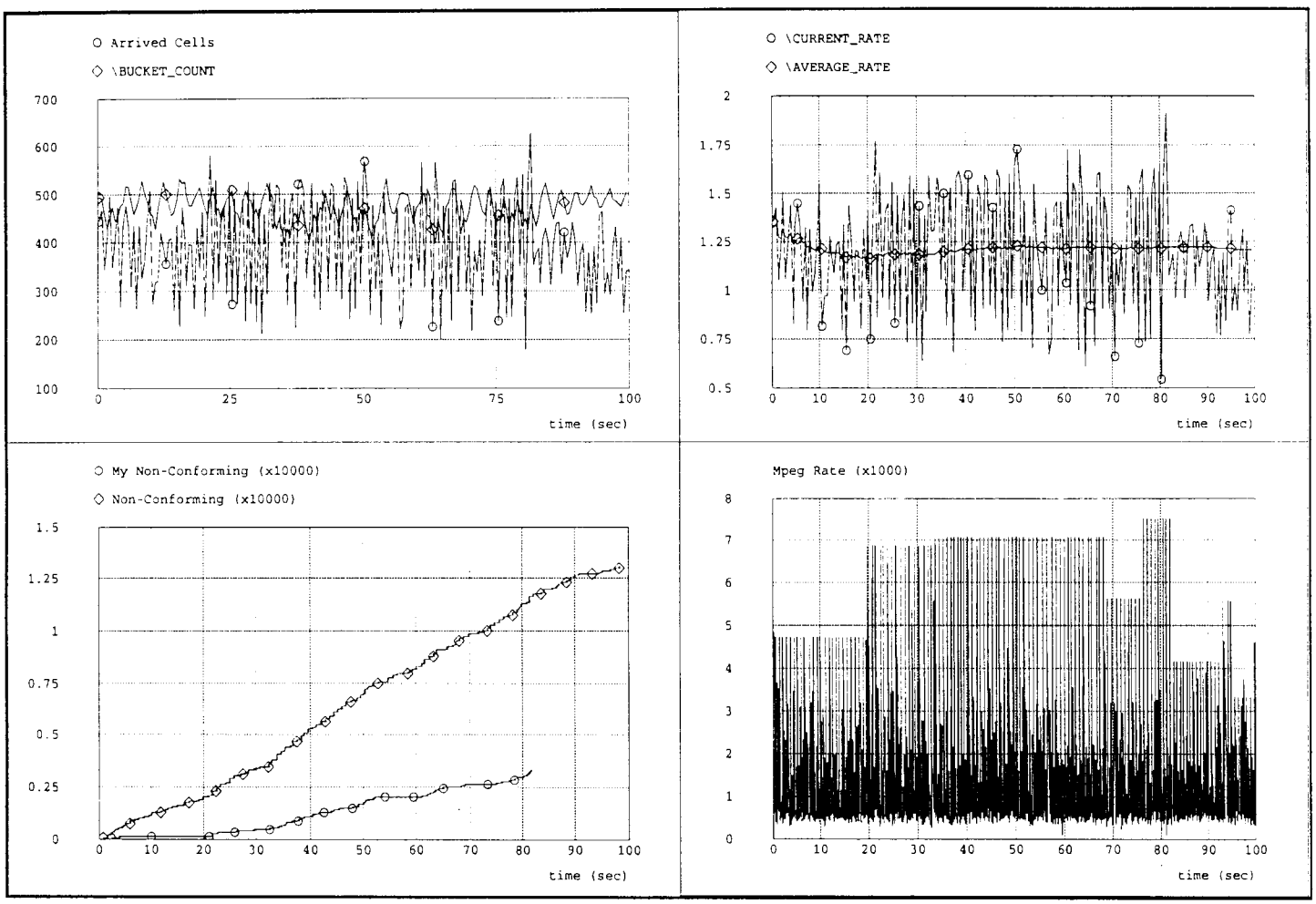

Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2.5
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
3
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

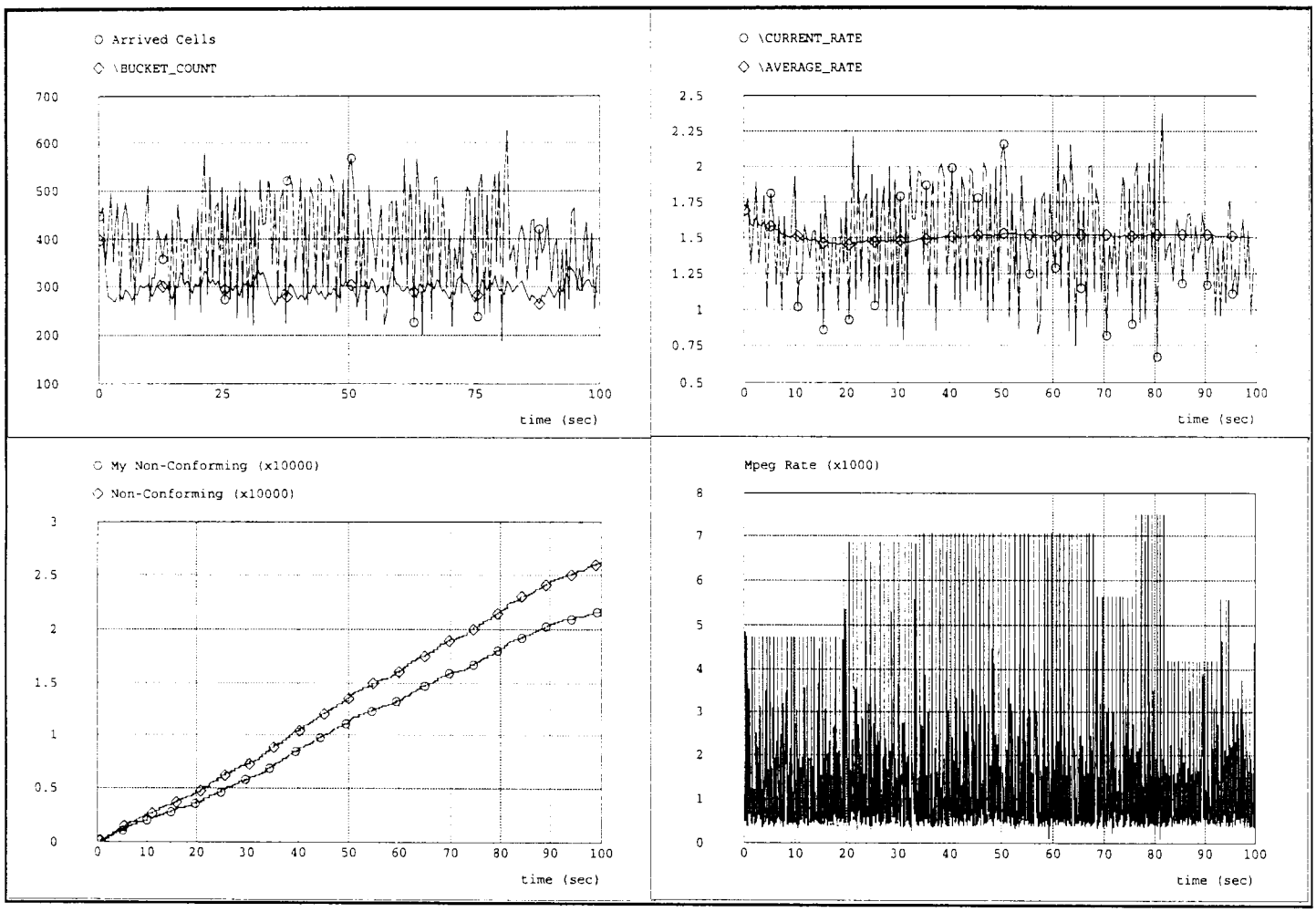

○ Arrived Cells (x1000)
◇ \BUCKET_COUNT (x1000)
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
time (sec)
Non-Conforming (x1000)
time (sec)
Mpeg Rate (x1000)
time (sec)

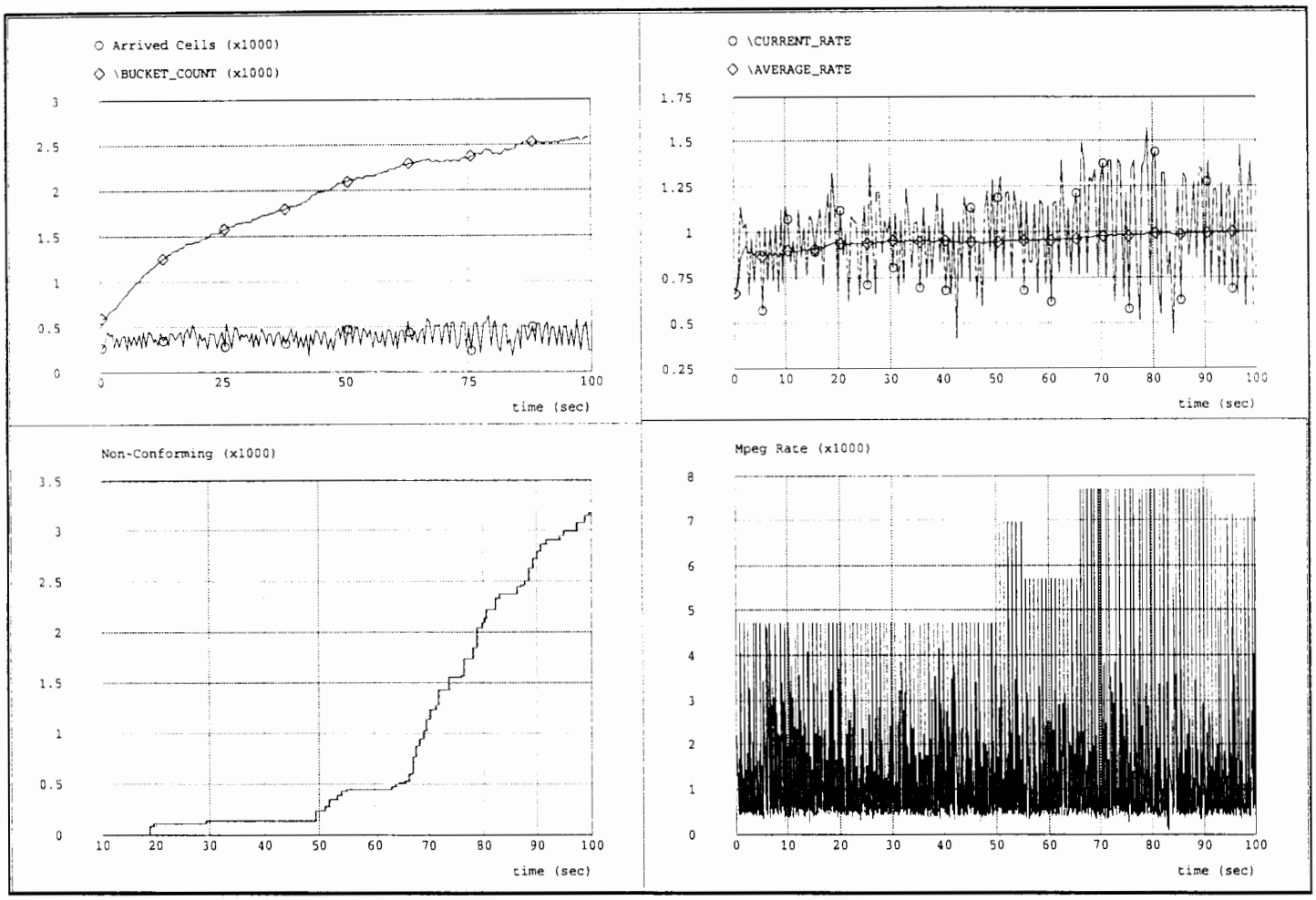

Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)

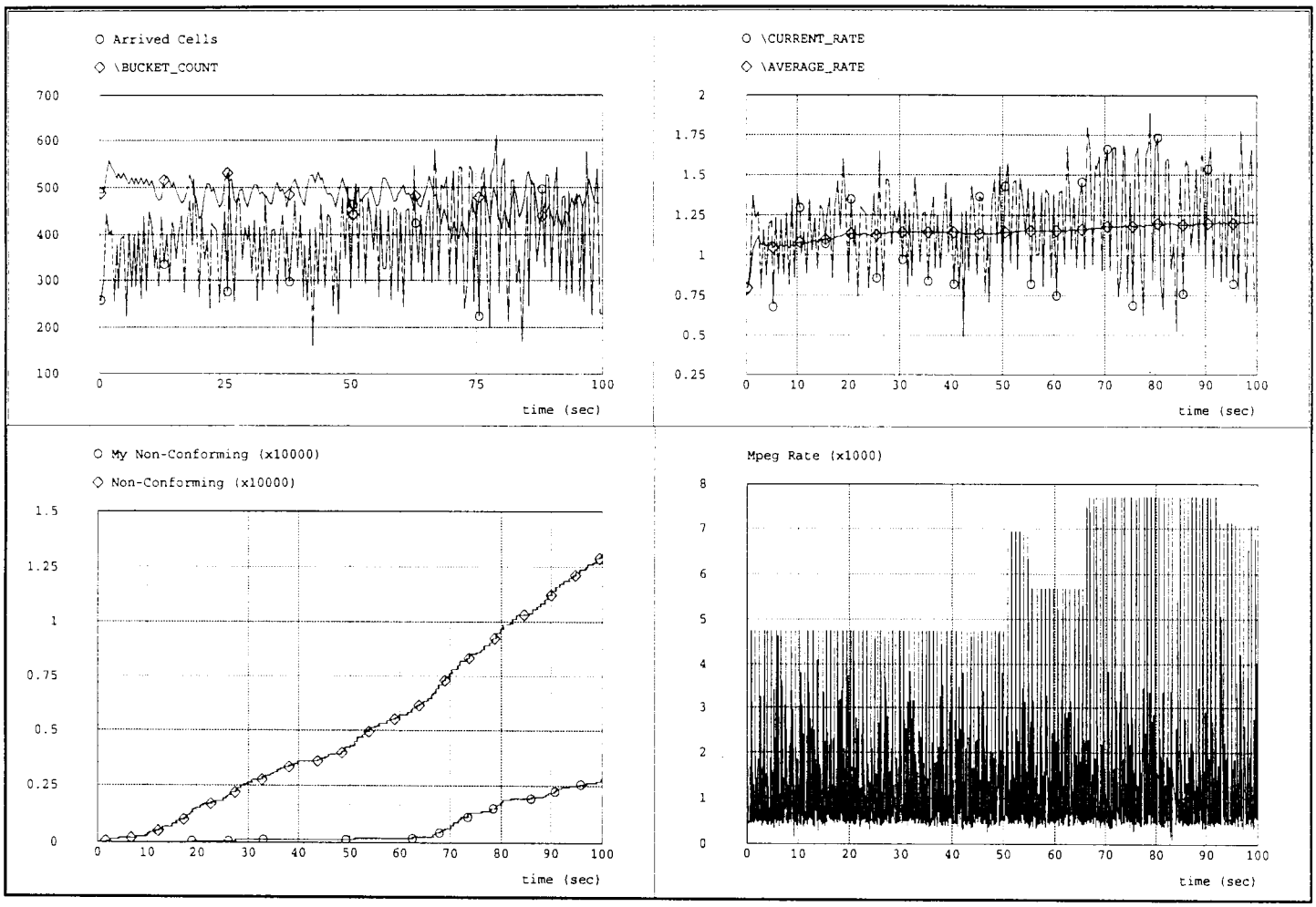

Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2.5
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
3
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

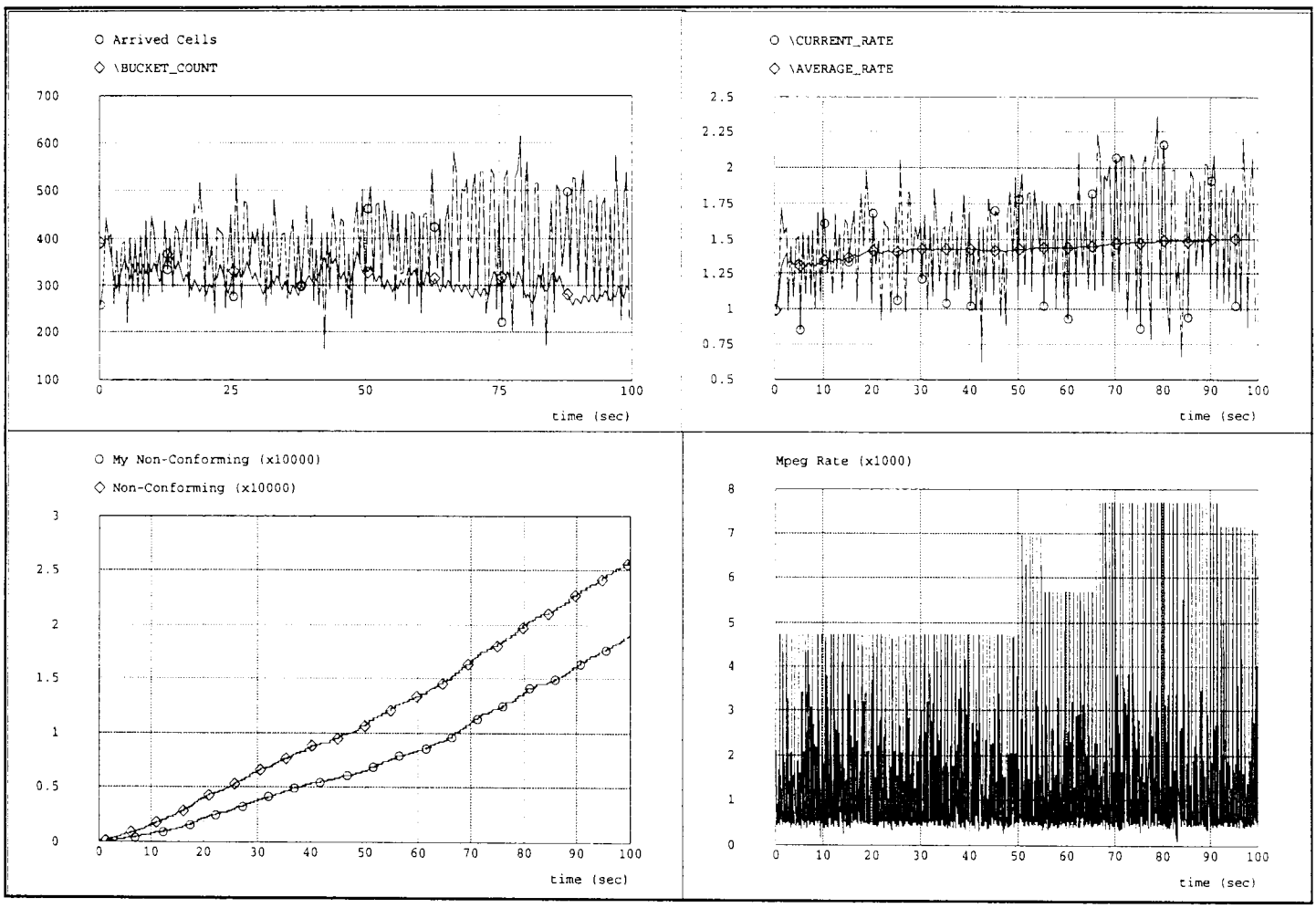

Arrived Cells (x1000)
\BUCKET_COUNT (x1000)
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
time (sec)
Non-Conforming (x1000)
time (sec)
Mpeg Rate (x1000)
time (sec)

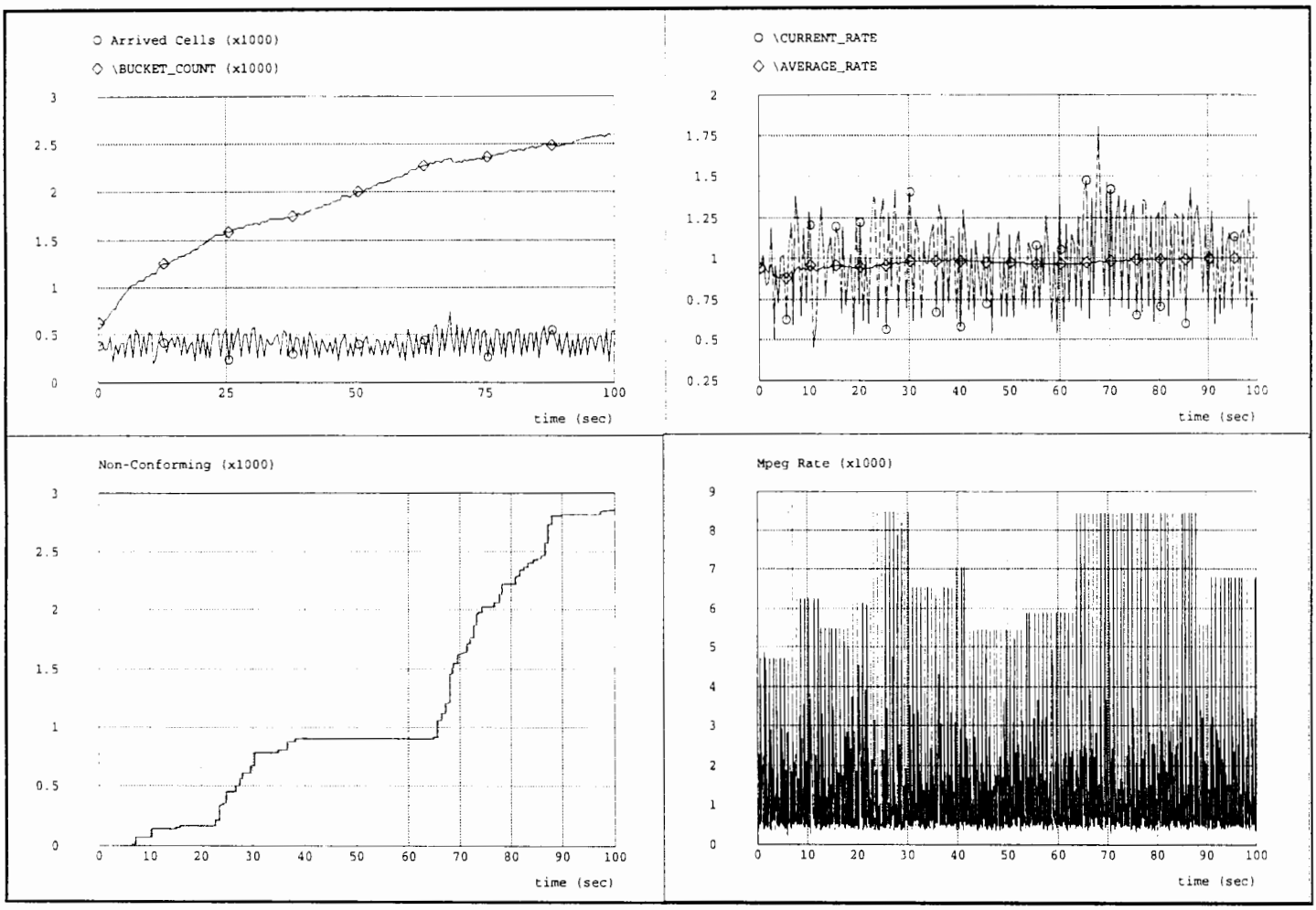

Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
1.5
1.25
1
0.75
0.5
0.25
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

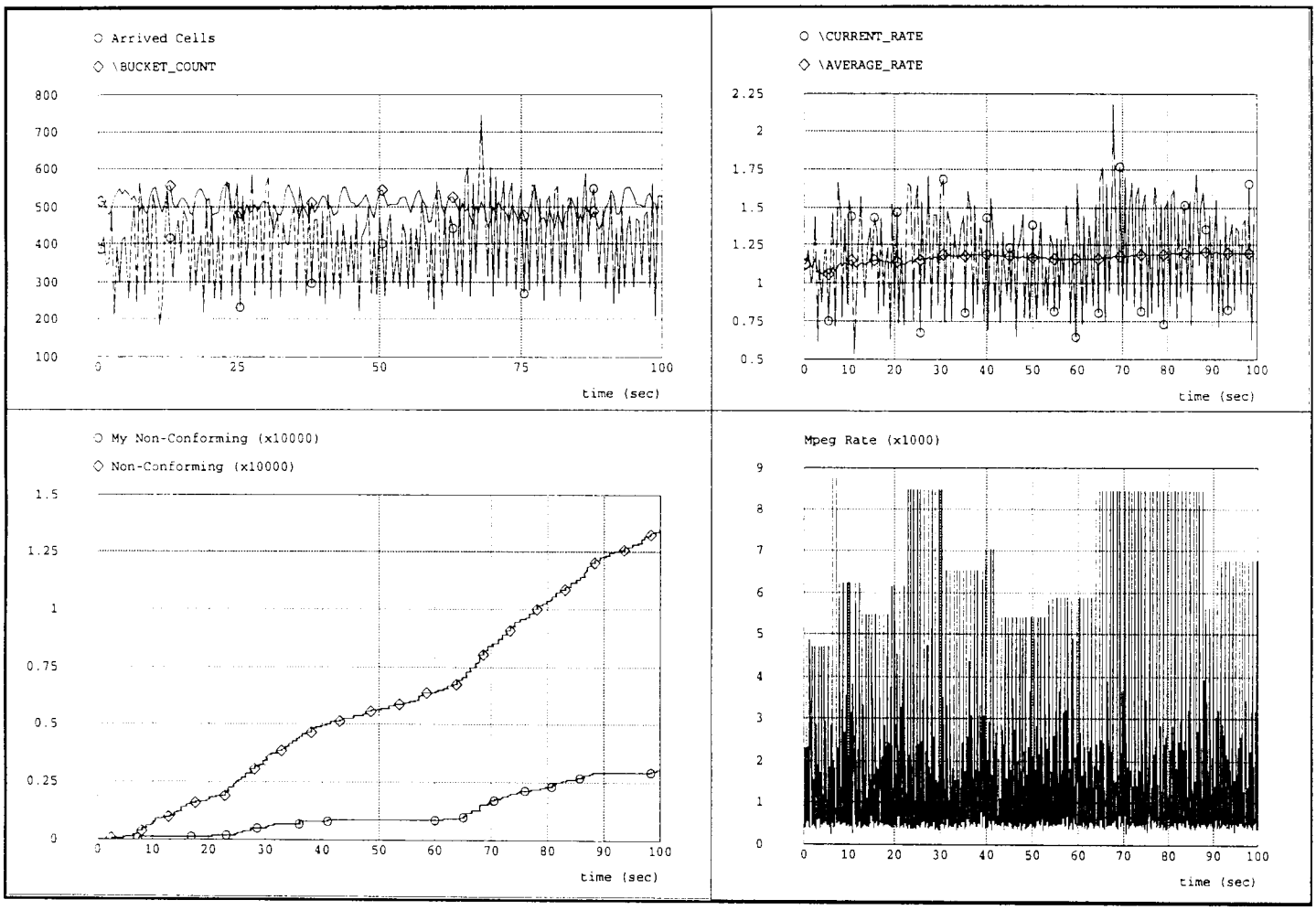

Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2.5
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
3
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)

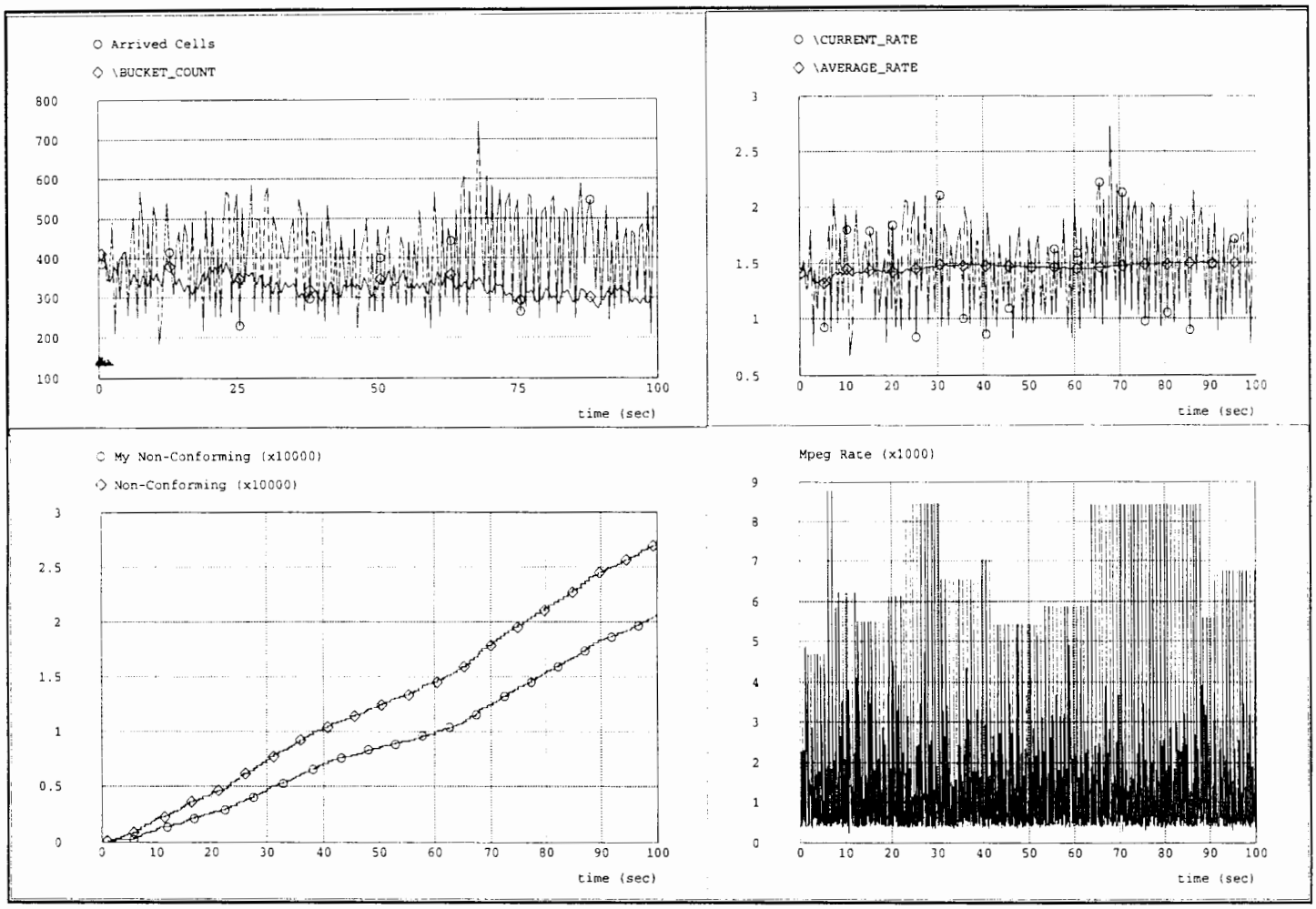

Arrived Cells (x1000)
\BUCKET_COUNT (x1000)
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
time (sec)
Non-Conforming (x1000)
time (sec)
Mpeg Rate (x1000)
time (sec)


Arrived Cells
\BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)


○ Arrived Cells
◇ \BUCKET_COUNT
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
○ \CURRENT_RATE
◇ \AVERAGE_RATE
2.5
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
○ My Non-Conforming (x10000)
◇ Non-Conforming (x10000)
3
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100
time (sec)


Arrived Cells (x1000)
\BUCKET_COUNT (x1000)
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
time (sec)
Non-Conforming (x1000)
time (sec)
Mpeg Rate (x1000)
time (sec)


Arrived Cells
\BUCKET_COUNT
800
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
1.5
1.25
1
0.75
0.5
0.25
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
9
8
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)


Arrived Cells
\BUCKET_COUNT
800
700
600
500
400
300
200
100
0
25
50
75
100
time (sec)
\CURRENT_RATE
\AVERAGE_RATE
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
My Non-Conforming (x10000)
Non-Conforming (x10000)
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
100
time (sec)
Mpeg Rate (x1000)
9
8
7
6
5
4
3
2
1
0
10
20
30
40
50
60
70
80
90
100
time (sec)

# APPENDIX D

## INSTALLING AND CONFIGURING FOR OPNET®

The simulations were done using OPNET® version 2.5b, and can be recreated by installing the simulation files from the disks that accompanies this dissertation. The installation process requires two machines, one running DOS and the other running UNIX, and it must be possible to FTP files from the DOS machine to the UNIX machine. The installation instructions follow.

## STEP 1: RESTORING FILES USING ARJ.

Create a temporary sub-directory on the DOS machine, and change to that sub-directory., for example

```
md c:\temp1234
c:
cd \temp1234
```

Use the ARJ compression program to decompress the source files from the accompanying disks into the temporary sub-directory. For example

```
arj x -va b:\thesis.arj
```
(b: is the drive on which thesis.arj (on first disk) can be found. Insert the second disk when prompted and press <Y> then <ENTER>.)

This will create a file named thesis.tar that has a file size of 4614144 bytes. This file has to be uploaded to the UNIX system.

## STEP 2: RESTORING FILES USING TAR.

On the UNIX system (where OPNET® resides) create the op_models sub-directory off the user's home directory, if it does not exists already. For example, if the user's home directory is /export/fred, then create /export/fred/op_models, and change to that sub-directory, e.g.

```
mkdir /export/fred/op_models
cd /export/fred/op_models
```

Use the DOS machine to upload the thesis.tar file into this sub-directory.

Restore thesis.tar using the UNIX tar command, for example

```
tar -xvf thesis.tar
```

A total of 54 files is restored, and are now available for the simulations. The thesis.tar file may be deleted if it still exists.

## STEP 3: RUNNING THE SIMULATIONS.

A script file (batch file) is provided that will run all thirty simulations and produce the output files needed for the OPNET® statistics. This file, final.run, executes the simulation file, final.sim, which produces the output files final1 to final30. One does not need the OPNET® environment to run the simulations in this way.

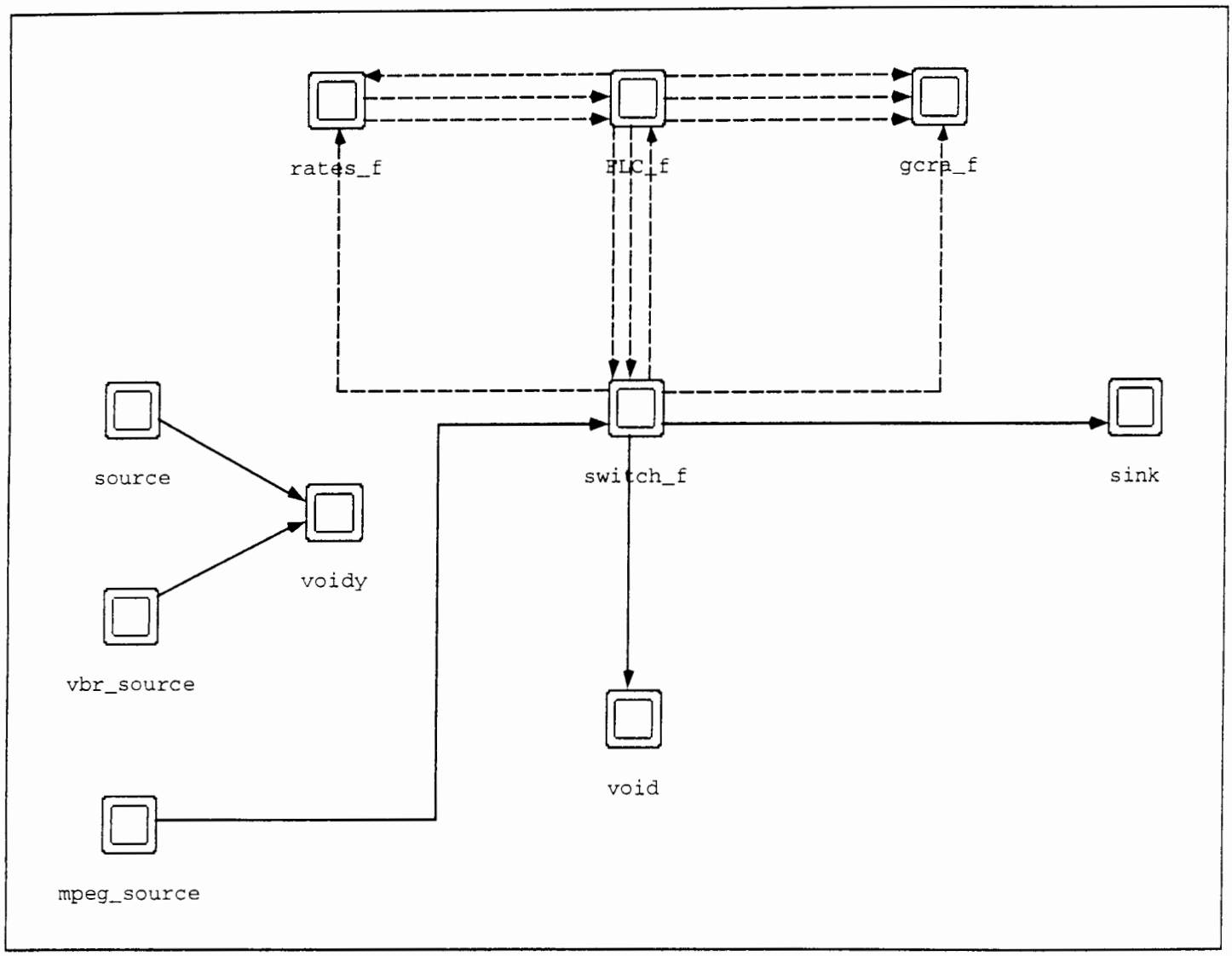
A different method is used when performing the simulations inside OPNET®. In the simulator load the final.seq file. This provides the information to run the thirty simulations.

If one needs to recompile the simulation, then load the final file into the network editor and initiate the 'recompile all' function.

# APPENDIX E

## DETAILED OPNET® FILES

# Fuzzy Logic Controller



...
...

| *processor* **source** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | source | string | p |
| process model | source_f | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | enabled | toggle | disabled |
| endsim intrpt | disabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | disabled | toggle | disabled |
| icon name | processor | icon | processor |
| source_rate | promoted | double | 1.0 |
| source_rate_dev | promoted | double | 1.0 |

| *packet stream* **source [0] -> voidy [1]** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | strm_5 | string | strm |
| src stream | 0 | enumerated | 0 |
| dest stream | 1 | enumerated | 0 |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| color | RGB030 | color | RGB030 |

| *processor* **switch_f** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | switch_f | string | p |
| process model | switch_f | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | disabled | toggle | disabled |
| endsim intrpt | disabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | disabled | toggle | disabled |
| icon name | processor | icon | processor |

| *packet stream* **switch_f [0] -> sink [0]** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | strm_1 | string | strm |
| src stream | 0 | enumerated | 0 |
| dest stream | 0 | enumerated | 0 |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| color | RGB030 | color | RGB030 |

| *statistic wire* **switch_f [outstat (2)] -> gcra_f [instat (3)]** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | stat_3 | string | stat |
| submodule | NONE | enumerated | NONE |

...
...

| | | | |
|---|---|---|---|
| src stat | outstat (2) | enumerated | outstat (0) |
| dest stat | instat (3) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

*statistic wire* **switch_f [outstat (0)] -> FLC_f [instat (0)]**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | stat_4 | string | stat |
| submodule | NONE | enumerated | NONE |
| src stat | outstat (0) | enumerated | outstat (0) |
| dest stat | instat (0) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

*statistic wire* **switch_f [outstat (1)] -> rates_f [instat (1)]**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | stat_7 | string | stat |
| submodule | NONE | enumerated | NONE |
| src stat | outstat (1) | enumerated | outstat (0) |
| dest stat | instat (1) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

*packet stream* **switch_f [1] -> void [0]**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | strm_2 | string | strm |
| src stream | 1 | enumerated | 0 |
| dest stream | 0 | enumerated | 0 |
| intrpt method | scheduled | integer | scheduled |

...
...

| | | | |
|---|---|---|---|
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| color | RGB030 | color | RGB030 |

***processor*** **sink**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | sink | string | p |
| process model | sink | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | disabled | toggle | disabled |
| endsim intrpt | disabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | disabled | toggle | disabled |
| icon name | processor | icon | processor |

***processor*** **FLC_f**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | FLC_f | string | p |
| process model | controller_f | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | enabled | toggle | disabled |
| endsim intrpt | enabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | enabled | toggle | disabled |
| icon name | processor | icon | processor |
| Negotiated_Peak_Rate | promoted | double | 1.0 |
| Burst_Size_GCRA | promoted | double | 1.0 |
| Initial_Burst_Size_FLC | promoted | double | 1.0 |
| Negotiated_Average_Rate | promoted | double | 1.0 |
| Window_Period_T | promoted | double | 1.0 |
| Token_Multiple | promoted | double | 2.0 |

***statistic wire*** **FLC_f [outstat (0)] -> gcra_f [instat (0)]**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | stat_0 | string | stat |
| submodule | NONE | enumerated | NONE |
| src stat | outstat (0) | enumerated | outstat (0) |
| dest stat | instat (0) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |

...
...

| | | | |
|---|---|---|---|
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

*statistic wire* **FLC_f [outstat (1)] -> gcra_f [instat (1)]**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | stat_1 | string | stat |
| submodule | NONE | enumerated | NONE |
| src stat | outstat (1) | enumerated | outstat (0) |
| dest stat | instat (1) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

*statistic wire* **FLC_f [outstat (2)] -> gcra_f [instat (2)]**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | stat_2 | string | stat |
| submodule | NONE | enumerated | NONE |
| src stat | outstat (2) | enumerated | outstat (0) |
| dest stat | instat (2) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

*statistic wire* **FLC_f [outstat (3)] -> switch_f [instat (1)]**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | stat_5 | string | stat |
| submodule | NONE | enumerated | NONE |
| src stat | outstat (3) | enumerated | outstat (0) |
| dest stat | instat (1) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

...
...

| *statistic wire* **FLC_f [outstat (4)] -> switch_f [instat (0)]** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | stat_6 | string | stat |
| submodule | NONE | enumerated | NONE |
| src stat | outstat (4) | enumerated | outstat (0) |
| dest stat | instat (0) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

| *statistic wire* **FLC_f [outstat (5)] -> rates_f [instat (0)]** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | stat_10 | string | stat |
| submodule | NONE | enumerated | NONE |
| src stat | outstat (5) | enumerated | outstat (0) |
| dest stat | instat (0) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

| *processor* **rates_f** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | rates_f | string | p |
| process model | rates_f | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | disabled | toggle | disabled |
| endsim intrpt | disabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | disabled | toggle | disabled |
| icon name | processor | icon | processor |

| *statistic wire* **rates_f [outstat (1)] -> FLC_f [instat (1)]** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | stat_8 | string | stat |

...
...

| | | | |
|---|---|---|---|
| submodule | NONE | enumerated | NONE |
| src stat | outstat (1) | enumerated | outstat (0) |
| dest stat | instat (1) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

*statistic wire* **rates_f [outstat (0)] -> FLC_f [instat (2)]**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | stat_9 | string | stat |
| submodule | NONE | enumerated | NONE |
| src stat | outstat (0) | enumerated | outstat (0) |
| dest stat | instat (2) | enumerated | instat (0) |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| rising edge trigger | disabled | toggle | enabled |
| falling edge trigger | disabled | toggle | enabled |
| repeated value trigger | disabled | toggle | disabled |
| zero crossing trigger | disabled | toggle | disabled |
| low threshold trigger | disabled | toggle double | disabled |
| high threshold trigger | -1.0 | toggle double | disabled |
| color | RGB331 | color | RGB331 |

*processor* **gcra_f**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | gcra_f | string | p |
| process model | gcra_f | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | disabled | toggle | disabled |
| endsim intrpt | disabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | disabled | toggle | disabled |
| icon name | processor | icon | processor |

*processor* **void**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | void | string | p |
| process model | sink | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | disabled | toggle | disabled |

...
...

| | | | |
|---|---|---|---|
| endsim intrpt | disabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | disabled | toggle | disabled |
| icon name | processor | icon | processor |

| *processor* **vbr_source** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | vbr_source | string | p |
| process model | catania_source | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | enabled | toggle | disabled |
| endsim intrpt | disabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | disabled | toggle | disabled |
| icon name | processor | icon | processor |
| mean_cells | promoted | double | 1.0 |
| mean_idle_time | promoted | double | 1.0 |
| inter_cell_time | promoted | double | 1.0 |

| *packet stream* **vbr_source [0] -> voidy [0]** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | strm_4 | string | strm |
| src stream | 0 | enumerated | 0 |
| dest stream | 0 | enumerated | 0 |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| color | RGB030 | color | RGB030 |

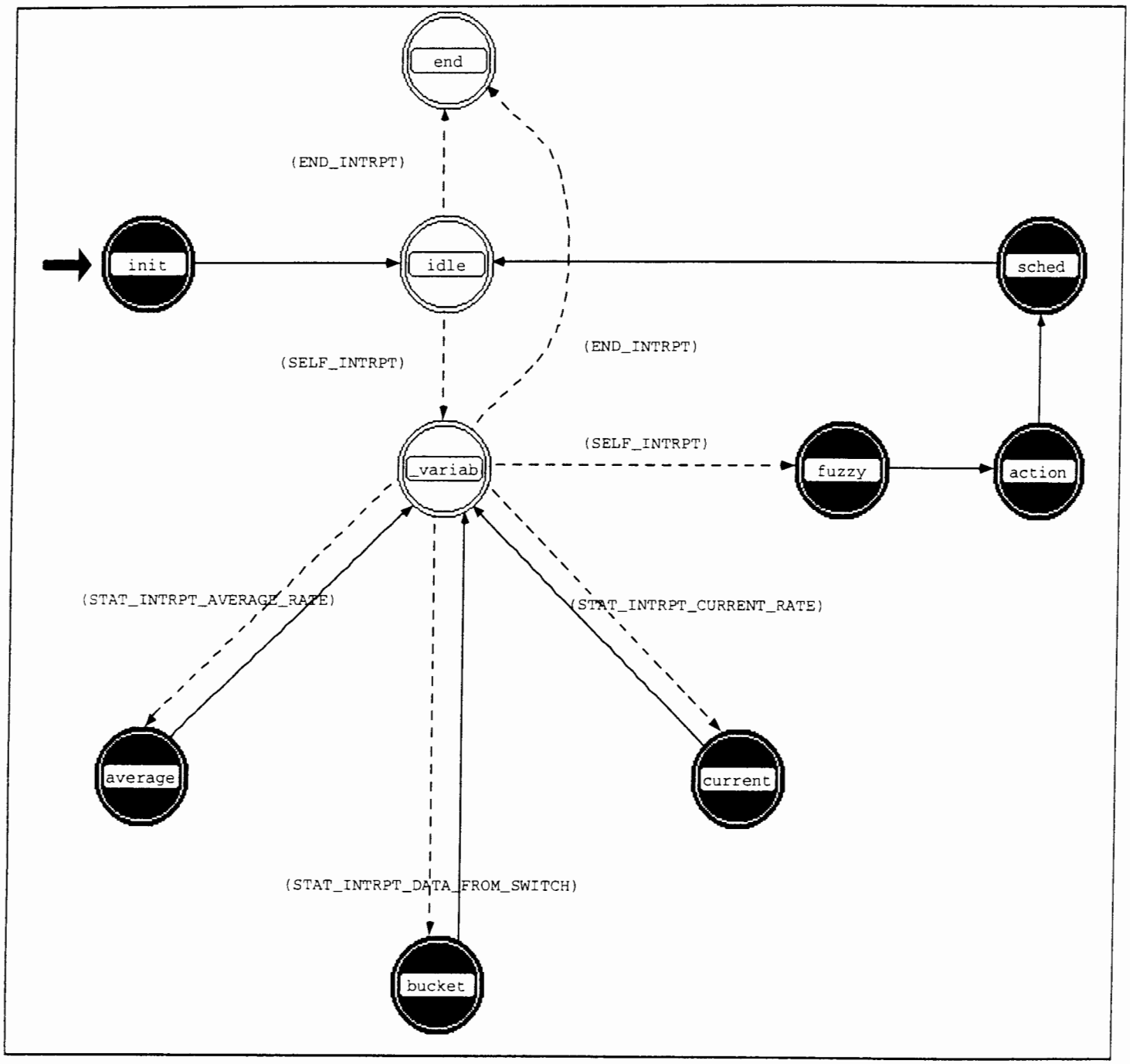
| *processor* **voidy** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | voidy | string | p |
| process model | sink | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | disabled | toggle | disabled |
| endsim intrpt | disabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | disabled | toggle | disabled |
| icon name | processor | icon | processor |

...
...

| *processor* **mpeg_source** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | mpeg_source | string | p |
| process model | mpeg_source | typed file | sink |
| intrpt interval | disabled | toggle double | disabled |
| begsim intrpt | enabled | toggle | disabled |
| endsim intrpt | disabled | toggle | disabled |
| failure intrpts | disabled | enumerated | disabled |
| recovery intrpts | disabled | enumerated | disabled |
| priority | 0 | integer | 0 |
| super priority | disabled | toggle | disabled |
| icon name | processor | icon | processor |

| *packet stream* **mpeg_source [0] -> switch_f [0]** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | strm_6 | string | strm |
| src stream | 0 | enumerated | 0 |
| dest stream | 0 | enumerated | 0 |
| intrpt method | scheduled | integer | scheduled |
| delay | 0.0 (sec.) | double | 0.0 (sec.) |
| color | RGB030 | color | RGB030 |

# Fuzzy Congestion Controller Process



...
...

## Header Block

```
#include "_opnet_.h"
#include "my_thesis.h"
#include "stdio.h"

#define SET_PEAK_RATE_GCRA              0
#define SET_AVERAGE_RATE_GCRA           1
#define SET_BURST_LENGTH_GCRA           2
#define ASK_FOR_DATA_FROM_SWITCH        3
#define SET_NUMBER_OF_CELLS_FOR_SWITCH  4
#define ASK_FOR_DATA_FROM_RATES         5
#define SET_BUCKET_SIZE                 6
#define SET_BUCKET_RATE                 7

#define CELLS_ARRIVED_FROM_SWITCH       0
#define AVERAGE_RATE_FROM_RATES         1
#define CURRENT_RATE_FROM_RATES         2

#define STAT_INTRPT_DATA_FROM_SWITCH    (STAT_INTRPT && op_intrpt_stat()== CELLS_ARRIVED_FROM_S
#define STAT_INTRPT_AVERAGE_RATE        (STAT_INTRPT && op_intrpt_stat()== AVERAGE_RATE_FROM_RATE
#define STAT_INTRPT_CURRENT_RATE        (STAT_INTRPT && op_intrpt_stat()== CURRENT_RATE_FROM_RATE

void do_plots(void);
```

## State Variable Block

```
double \ALL_OK;
FILE *\file_p;

double \Window_T;
double \burst_size_gcra, \burst_size_flc;
double \negotiated_peak_rate, \negotiated_average_rate;
double \token_multiple;

double \time, \bucket_count;
double \decision;
double \rates_current_rate;
double \rates_average_rate;
double \arrived_cells;


int \got_all_three_inputs;
```

## Temporary Variable Block

```
double t;
Objid id;
int test;
double Na, Np;
double p_over_a;
```

## Function Block

```
/*
    This Code is too long to print.
  It can be found on the disk that
```

...
...

```
    accompanies this thesis.
*/
```

**Diagnostic Block**

```
/*#define DEBUG_INFO
*/
```

***forced state*** **init**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | init | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

***enter execs*** **init**

```
id=op_id_self();

/*get the simulation variables*/
op_ima_obj_attr_get(id,"Window_Period_T", &Window_T);
op_ima_obj_attr_get(id,"Burst_Size_GCRA", &burst_size_gcra);
op_ima_obj_attr_get(id,"Initial_Burst_Size_FLC", &burst_size_flc);
op_ima_obj_attr_get(id,"Negotiated_Peak_Rate", &negotiated_peak_rate);
op_ima_obj_attr_get(id,"Negotiated_Average_Rate", &negotiated_average_rate);
op_ima_obj_attr_get(id,"Token_Multiple", &token_multiple);

ALL_OK=0;

/*initialise rates counter*/
op_stat_local_write(ASK_FOR_DATA_FROM_RATES, 2);

/*initialise the number of cells that the switch must allow through*/
op_stat_local_write(SET_NUMBER_OF_CELLS_FOR_SWITCH, burst_size_flc);

/*initialse GCRA*/
op_stat_local_write(SET_PEAK_RATE_GCRA, negotiated_peak_rate);
op_stat_local_write(SET_AVERAGE_RATE_GCRA, negotiated_average_rate);
op_stat_local_write(SET_BURST_LENGTH_GCRA, burst_size_gcra);

/*initialse Leaky Bucket*/
/*
    op_stat_local_write(SET_BUCKET_SIZE, burst_size_gcra);
    op_stat_local_write(SET_BUCKET_RATE, negotiated_average_rate);
*/
time=op_sim_time();
rates_current_rate=0.0;
rates_average_rate=0.0;
got_all_three_inputs=0;
arrived_cells=0;
bucket_count=0;

/*let the Fuzzy Logic Controller know about the OPNET variables it needs*/
op_register_state_variable("BUCKET_COUNT", &(bucket_count));
```

...
...

```
    /*op_register_state_variable("ARRIVED_CELLS", &(arrived_cells));*/
    op_register_state_variable("DECISION", &(decision));
    op_register_state_variable("TIME", &(time));
    op_register_state_variable("CURRENT_RATE", &(rates_current_rate));
    op_register_state_variable("AVERAGE_RATE", &(rates_average_rate));

    /*initialise fuzzy logic engine*/
    printf("Here.\n");
    old_fuzzy_init();

    /*modify sets*/
    p_over_a=negotiated_peak_rate/negotiated_average_rate;
    Np=negotiated_peak_rate*Window_T;
    Na=negotiated_average_rate*Window_T;

    op_change_fuzzy_var_range("CCR", 0.0, p_over_a);
    op_change_fuzzy_var_range("ACR", 0.0, p_over_a);
    op_change_fuzzy_var_range("BUCKET", 0.0, Na*token_multiple);
    /*arrived_cells=Na;  */
    bucket_count=burst_size_flc;
    op_change_fuzzy_var_init("BUCKET", bucket_count);
    op_change_fuzzy_var_init("ACR", 1.0);
    op_change_fuzzy_var_init("CCR", 1.0);

    if (p_over_a>1.5)
    {
        printf("First set of sets used: Peak/Average > 1.5.\n");

        op_change_fuzzy_set("CCR", "H",
                            1.0, 0.0,
                            p_over_a/**Window_T*/, 1.0,
                            p_over_a/**Window_T*/, 0.0,
                            p_over_a/**Window_T*/, 0.0);

        op_change_fuzzy_set("ACR", "H",
                            1.0, 0.0,
                            p_over_a/**Window_T*/, 1.0,
                            p_over_a/**Window_T*/, 0.0,
                            p_over_a/**Window_T*/, 0.0);

        op_change_fuzzy_set("BUCKET", "L",
                            0.0, 1.0,
                            Na, 0.0,
                            Na, 0.0,
                            Na, 0.0);
        op_change_fuzzy_set("BUCKET", "M",
                            0.5*Na, 0.0,
                            Na, 1.0,
                            1.5*Na, 0.0,
                            1.5*Na, 0.0);
        op_change_fuzzy_set("BUCKET", "H",
                            Na, 0.0,
                            Np, 1.0,
                            Na*token_multiple, 1.0,
                            Na*token_multiple, 0.0);
    }
    else
    {
        op_change_fuzzy_set("CCR", "M",
```

...
...

```
                                0.5, 0.0,
                                1.0, 1.0,
                                p_over_a, 0.0,
100                             p_over_a, 0.0);
        op_change_fuzzy_set("CCR", "H",
                                1.0, 0.0,
                                p_over_a, 1.0,
                                p_over_a, 0.0,
105                             p_over_a, 0.0);

        op_change_fuzzy_set("ACR", "M",
                                0.5, 0.0,
                                1.0, 1.0,
110                             p_over_a, 0.0,
                                p_over_a, 0.0);
        op_change_fuzzy_set("ACR", "H",
                                1.0, 0.0,
                                p_over_a, 1.0,
115                             p_over_a, 0.0,
                                p_over_a, 0.0);

        op_change_fuzzy_set("BUCKET", "L",
                                0.0, 1.0,
120                             Na, 0.0,
                                Na, 0.0,
                                Na, 0.0);
        op_change_fuzzy_set("BUCKET", "M",
                                0.5*Na, 0.0,
125                             Na, 1.0,
                                Np, 0.0,
                                Np, 0.0);
        op_change_fuzzy_set("BUCKET", "H",
                                Na, 0.0,
130                             Np, 1.0,
                                Na*token_multiple, 1.0,
                                Na*token_multiple, 0.0);
    }

135 op_change_fuzzy_set("SA", "NB",
                            -3.0*Na/9, 1.0,
                            -2.0*Na/9, 1.0,
                            -Na/9, 0.0,
                            -Na/9, 0.0);
140 op_change_fuzzy_set("SA", "NS",
                            -2.0*Na/9, 0.0,
                            -Na/9, 1.0,
                            0.0, 0.0,
                            0.0, 0.0);
145 op_change_fuzzy_set("SA", "Z",
                            -Na/9, 0.0,
                            0.0, 1.0,
                            Na/9, 0.0,
                            Na/9, 0.0);
150 op_change_fuzzy_set("SA", "PS",
                            0.0, 0.0,
                            Na/9, 1.0,
                            2.0*Na/9, 0.0,
                            2.0*Na/9, 0.0);
155 op_change_fuzzy_set("SA", "PB",
```

...
...

```
                    Na/9, 0.0,
                    2.0*Na/9, 1.0,
                    3.0*Na/9, 1.0,
                    3.0*Na/9, 1.0);
160 op_change_fuzzy_var_range("SA", -3.0*Na/9, 3.0*Na/9);

    /*prepare to interrupt process after the Window period*/
    op_intrpt_schedule_self(op_sim_time()+Window_T,0);
```

***transition*** **init -> idle**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_33 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***unforced state*** **get_variables**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | get_variables | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | unforced | toggle | unforced |

***enter execs*** **get_variables**

```
    if (got_all_three_inputs==3)
    {
        /*prepare to interrupt process:*/
        /*done with variable collection, so continue immediately*/
 5      op_intrpt_schedule_self(op_sim_time()/*+Window_T*/,0);

        time=op_sim_time();

        /*printf("Got them\n");  */
10  }
```

***transition*** **get_variables -> bucket**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_49 | string | tr |
| condition | STAT_INTRPT_DATA_FR... | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***transition*** **get_variables -> average**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_52 | string | tr |

...
...

| | | | |
|---|---|---|---|
| condition | STAT_INTRPT_AVERAGE... | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***transition*** **get_variables -> current**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_53 | string | tr |
| condition | STAT_INTRPT_CURRENT... | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***transition*** **get_variables -> fuzzy**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_55 | string | tr |
| condition | SELF_INTRPT | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***transition*** **get_variables -> end**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_62 | string | tr |
| condition | END_INTRPT | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***forced state*** **sched**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | sched | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

***enter execs*** **sched**

```
/*prepare to interrupt process after the Window period*/
op_intrpt_schedule_self(op_sim_time()+Window_T,0);
```

***transition*** **sched -> idle**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_59 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

...
...

*forced state* **fuzzy**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | fuzzy | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

*enter execs* **fuzzy**

```
    if (ALL_OK)
    {
        fuzzify();
        defuzzify();
5       do_plots();
    }
```

*transition* **fuzzy -> action**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_56 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

*forced state* **action**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | action | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

*enter execs* **action**

```
    /*
      change the bucket count, the number of tokens the source has earned,
      by the controller decision
    */
5   bucket_count+=decision;

    /*make sure it is within range*/
    if (bucket_count<0.0)
        bucket_count=0.0;
10  if (bucket_count>(negotiated_average_rate*Window_T*token_multiple))
        bucket_count=negotiated_average_rate*Window_T*token_multiple;

    /*and inform the switch of the change*/
    op_stat_local_write(SET_NUMBER_OF_CELLS_FOR_SWITCH, bucket_count);
```

...
...

*transition* **action -> sched**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_58 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

*unforced state* **end**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | end | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | unforced | toggle | unforced |

*enter execs* **end**

```
fclose(file_p);
print_all_sets();
```

*forced state* **current**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | current | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

*enter execs* **current**

```
got_all_three_inputs++;
rates_current_rate=op_stat_local_read(CURRENT_RATE_FROM_RATES)/negotiated_average_rate;
/*printf("Got Current\n");    */
```

*transition* **current -> get_variables**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_47 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

*forced state* **average**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | average | string | st |

...
...

| | | | |
|---|---|---|---|
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

***enter execs*** **average**

```
got_all_three_inputs++;
rates_average_rate=op_stat_local_read(AVERAGE_RATE_FROM_RATES)/negotiated_average_rate;
/*printf("Got Average\n");  */
```

***transition*** **average -> get_variables**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_43 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***unforced state*** **idle**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | idle | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (See below.) | textlist | (See below.) |
| status | unforced | toggle | unforced |

***enter execs*** **idle**

***exit execs*** **idle**

```
got_all_three_inputs=0;
/*printf("Waiting for 3 inputs.\n");*/
op_stat_local_write(ASK_FOR_DATA_FROM_SWITCH, 2);
op_stat_local_write(ASK_FOR_DATA_FROM_RATES, 2);
```

***transition*** **idle -> end**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_36 | string | tr |
| condition | END_INTRPT | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***transition*** **idle -> get_variables**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_54 | string | tr |
| condition | SELF_INTRPT | string | |

...
...

| | | | |
|---|---|---|---|
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***forced state*** **bucket**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | bucket | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

***enter execs*** **bucket**

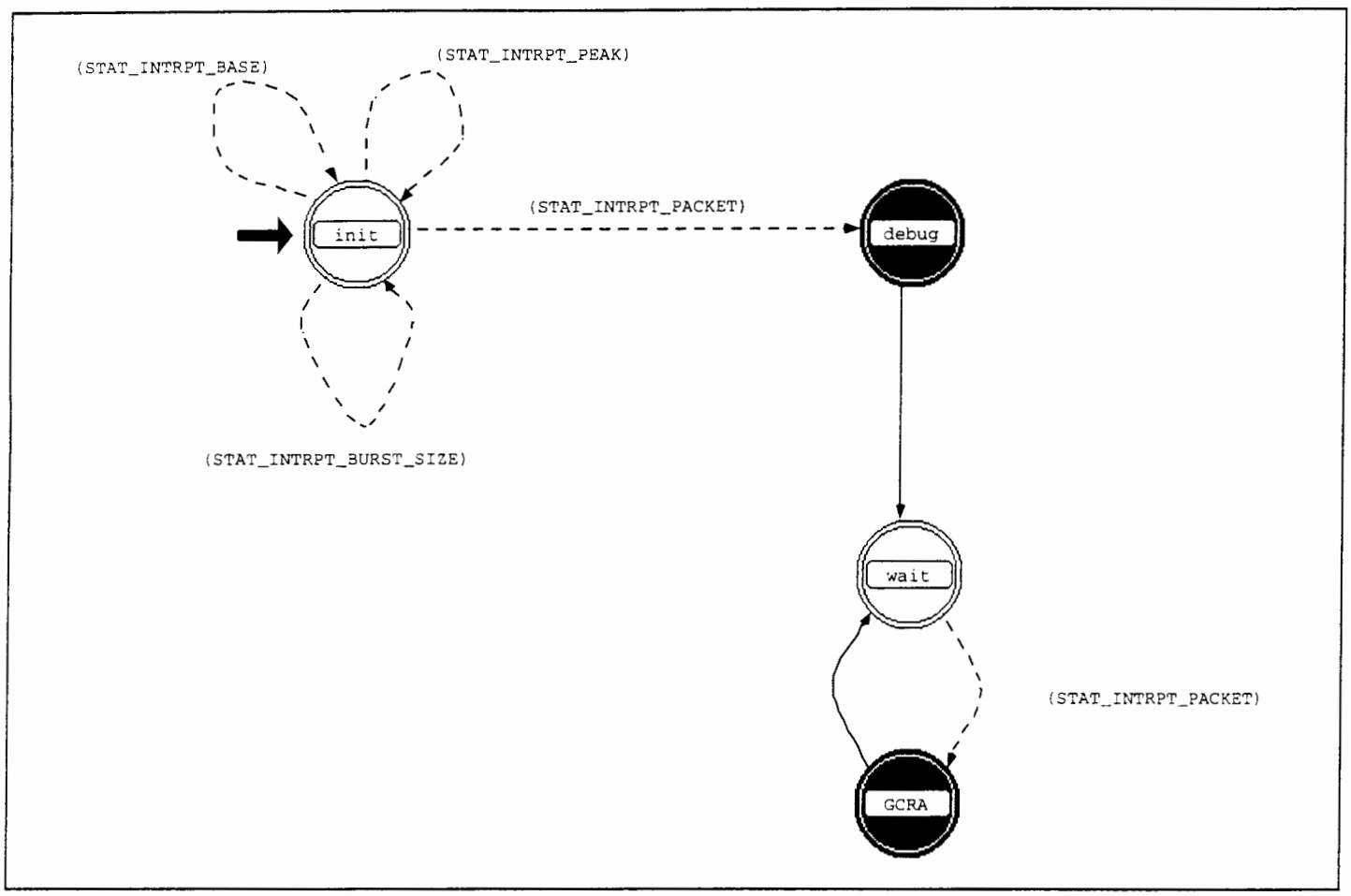
```
5
got_all_three_inputs++;
arrived_cells=op_stat_local_read(CELLS_ARRIVED_FROM_SWITCH);

/*printf("Got Switch stuff\n"); */
```

***transition*** **bucket -> get_variables**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_48 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

# GCRA Process



...
...

## Header Block

```
#include "my_thesis.h"

#define BASE_STAT                1
#define PEAK_STAT                0
#define PACKET_STAT              3
#define BURST_SIZE_STAT          2

#define STAT_INTRPT_BASE         (STAT_INTRPT && op_intrpt_stat()== BASE_STAT)
#define STAT_INTRPT_PEAK         (STAT_INTRPT && op_intrpt_stat()== PEAK_STAT)
#define STAT_INTRPT_BURST_SIZE   (STAT_INTRPT && op_intrpt_stat()== BURST_SIZE_STAT)
#define STAT_INTRPT_PACKET       (STAT_INTRPT && op_intrpt_stat()== PACKET_STAT)
```

## State Variable Block

```
double \TAT_peak;
double \L_peak;
double \I_peak;
double \TAT_sustain;
double \L_sustain;
double \I_sustain;
double \conforming, \non_conforming;
double \tc, \tn;
double \base_gen, \peak_rate;
double \burst_size;
```

## Temporary Variable Block

```
double ta;
Objid id;
int is_conforming;
```

## *unforced state* init

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | init | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | unforced | toggle | unforced |

## *enter execs* init

```
printf("In GCRA init.\n");

if (op_intrpt_stat()==BASE_STAT)
{
    base_gen=op_stat_local_read(BASE_STAT);
    printf("  Read Ave Rate=%4.4lf,\nPeak Rate=%4.4lf.\n",
        base_gen, peak_rate);
}
else if (op_intrpt_stat()==PEAK_STAT)
{
    peak_rate=op_stat_local_read(PEAK_STAT);
    printf("  Read Peak Rate=%4.1lf.\n", peak_rate);
```

...
...

```
}
else if (op_intrpt_stat()==BURST_SIZE_STAT)
{
    burst_size=op_stat_local_read(BURST_SIZE_STAT);
    printf("  Read Burst size=%4.1lf.\n", burst_size);
}

ta=op_sim_time();
TAT_peak=ta;
TAT_sustain=ta;
if (peak_rate)
    I_peak=1/peak_rate;
if (base_gen)
    I_sustain=1/base_gen;
L_peak=0;
L_sustain=(burst_size-1)*(I_sustain-I_peak);
conforming=0.0;
non_conforming=0.0;
tc=tn=0.0;
```

| *transition* **init -> init** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | tr_8 | string | tr |
| condition | STAT_INTRPT_BASE | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

| *transition* **init -> init** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | tr_9 | string | tr |
| condition | STAT_INTRPT_PEAK | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

| *transition* **init -> init** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | tr_10 | string | tr |
| condition | STAT_INTRPT_BURST_S... | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

| *transition* **init -> debug** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | tr_11 | string | tr |
| condition | STAT_INTRPT_PACKET | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

...
...

***unforced state*** **wait**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | wait | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | unforced | toggle | unforced |

***enter execs*** **wait**

```
/*("Cell arrived ...");*/
/*("In GCRA wait.\n");*/
```

***transition*** **wait -> GCRA**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_2 | string | tr |
| condition | STAT_INTRPT_PACKET | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***forced state*** **GCRA**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | GCRA | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

***enter execs*** **GCRA**

```
/* The Generic Cell Rate Algorithm */

ta=op_sim_time();
is_conforming=0;
if (TAT_peak<ta)
{
    TAT_peak=ta;
    TAT_peak=TAT_peak+I_peak;
    tc++;
    is_conforming=1;
}
else
{
    if (TAT_peak>(ta+L_peak))
    {  non_conforming++;
       tn++;

       op_stat_global_write(op_stat_global_reg("Non-Conforming"), non_conforming);
    }
    else
    {
```

...
...

```
            TAT_peak=TAT_peak+I_peak;
            is_conforming=1;
        }
}
if (is_conforming)
{
    if (TAT_sustain<ta)
    {
        TAT_sustain=ta;
        TAT_sustain=TAT_sustain+I_sustain;
        conforming++;
        op_stat_global_write(op_stat_global_reg("Conforming"), conforming);
    }
    else
    {
        if (TAT_sustain>(ta+L_sustain))
        {  non_conforming++;
            op_stat_global_write(op_stat_global_reg("Non-Conforming"), non_conforming);
        }
        else
        {
            TAT_sustain=TAT_sustain+I_sustain;
            conforming++;
            op_stat_global_write(op_stat_global_reg("Conforming"), conforming);

        }
    }
}
```

***transition*** **GCRA -> wait**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_3 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***forced state*** **debug**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | debug | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

***enter execs*** **debug**

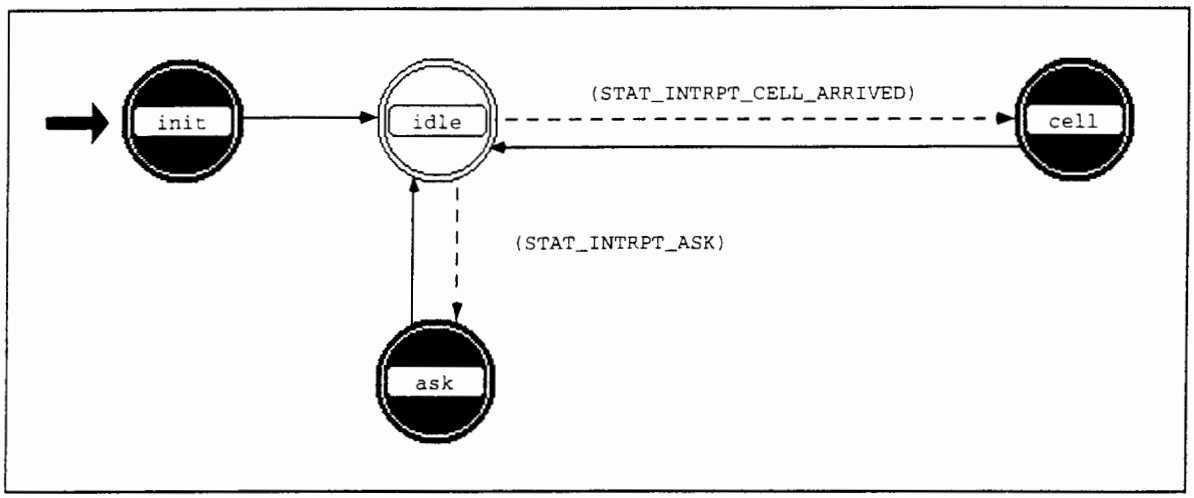
```
printf("Going to GCRA.\n");
printf("    TAT_peak=%4.4lf\n", TAT_peak);
printf("    I_peak=%4.4lf\n", I_peak);
printf("    L_peak=%4.4lf\n", L_peak);
printf("    TAT_sustain=%4.4lf\n", TAT_sustain);
printf("    I_sustain=%4.4lf\n", I_sustain);
```

...
...

```
printf("      L_sustain=%4.4lf\n", L_sustain);
```

| ***transition*** **debug -> wait** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | tr_12 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

# Rate Counter Process



...
...

*forced state* **ask**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | ask | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

*enter execs* **ask**

```
/*
     Controller asked for data.
*/

op_stat_local_write(CURRENT_RATE_TO_FLC,
     number_of_cells_in_window/(op_sim_time()-window_start_time));
op_stat_local_write(AVERAGE_RATE_TO_FLC,
     total_number_of_cells/op_sim_time());
number_of_cells_in_window=0;
window_start_time=op_sim_time();
```

*transition* **ask -> idle**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_4 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

...
...

## Header Block

```
#include "my_thesis.h"

#define CURRENT_RATE_TO_FLC        0
#define AVERAGE_RATE_TO_FLC        1

#define ASK_FOR_DATA               0
#define CELL_ARRIVED               1
#define RESET                      2

#define STAT_INTRPT_CELL_ARRIVED (STAT_INTRPT && op_intrpt_stat()== CELL_ARRIVED)
#define STAT_INTRPT_ASK          (STAT_INTRPT && op_intrpt_stat()== ASK_FOR_DATA)
#define STAT_INTRPT_RESET        (STAT_INTRPT && op_intrpt_stat()== RESET)
```

## State Variable Block

```
/*number of cells since start of connection*/
double \total_number_of_cells;

/*number of cells since start of window*/
double \number_of_cells_in_window;

/*start time of current window */
double \window_start_time;
```

## Temporary Variable Block

```
double time;
```

## *forced state* init

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | init | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

## *enter execs* init

```
total_number_of_cells=0;
number_of_cells_in_window=0;
window_start_time=op_sim_time();
/*printf("Rates initiated.\n");*/
```

## *transition* init -> idle

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_0 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

...
...

***unforced state*** **idle**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | idle | string | st |
| enter execs | (empty) | textlist | (empty) |
| exit execs | (empty) | textlist | (empty) |
| status | unforced | toggle | unforced |

***transition*** **idle -> cell**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_1 | string | tr |
| condition | STAT_INTRPT_CELL_AR... | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***transition*** **idle -> ask**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_3 | string | tr |
| condition | STAT_INTRPT_ASK | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***forced state*** **cell**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | cell | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

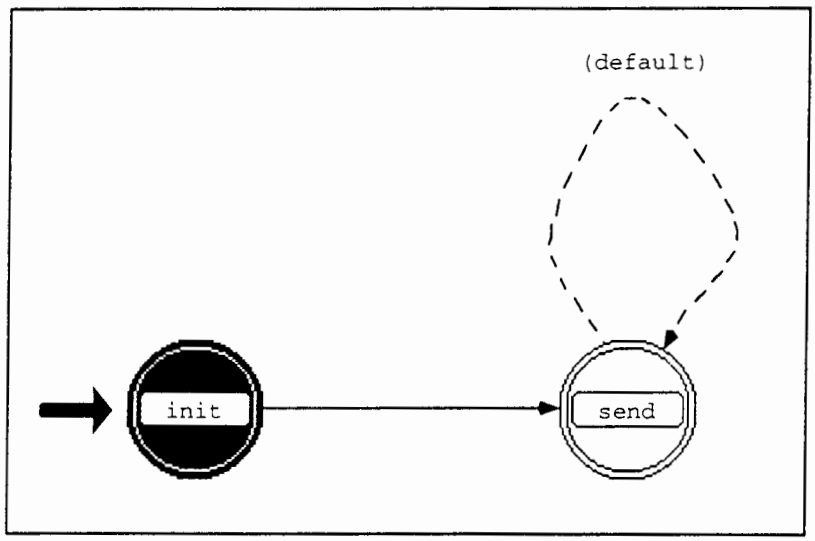
***enter execs*** **cell**

```
total_number_of_cells++;
number_of_cells_in_window++;
op_stat_global_write(op_stat_global_reg("Number of Cells"), total_number_of_cells);
```

***transition*** **cell -> idle**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_2 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

# Simple Source



...
...

**State Variable Block**

```
Distribution *\dist;
double \source_rate;
double \source_rate_dev;
```

**Temporary Variable Block**

```
Objid id;
double next_arrival_time;
Packet *pkt;
double d;
```

*forced state* **init**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | init | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

*enter execs* **init**

```
id=op_id_self();

/* Get the parametera */
op_ima_obj_attr_get(id,"source_rate", &source_rate);
op_ima_obj_attr_get(id,"source_rate_dev", &source_rate_dev);

/*load normal distribution */
d=source_rate_dev/(source_rate*(source_rate+source_rate_dev));
dist=op_dist_load("normal", 1/source_rate, d*d);

```

*transition* **init -> send**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_0 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

*unforced state* **send**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | send | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | unforced | toggle | unforced |

*enter execs* **send**

```
/*predict next cell time and schedule it*/
while ((next_arrival_time=op_dist_outcome(dist))<=0);
```

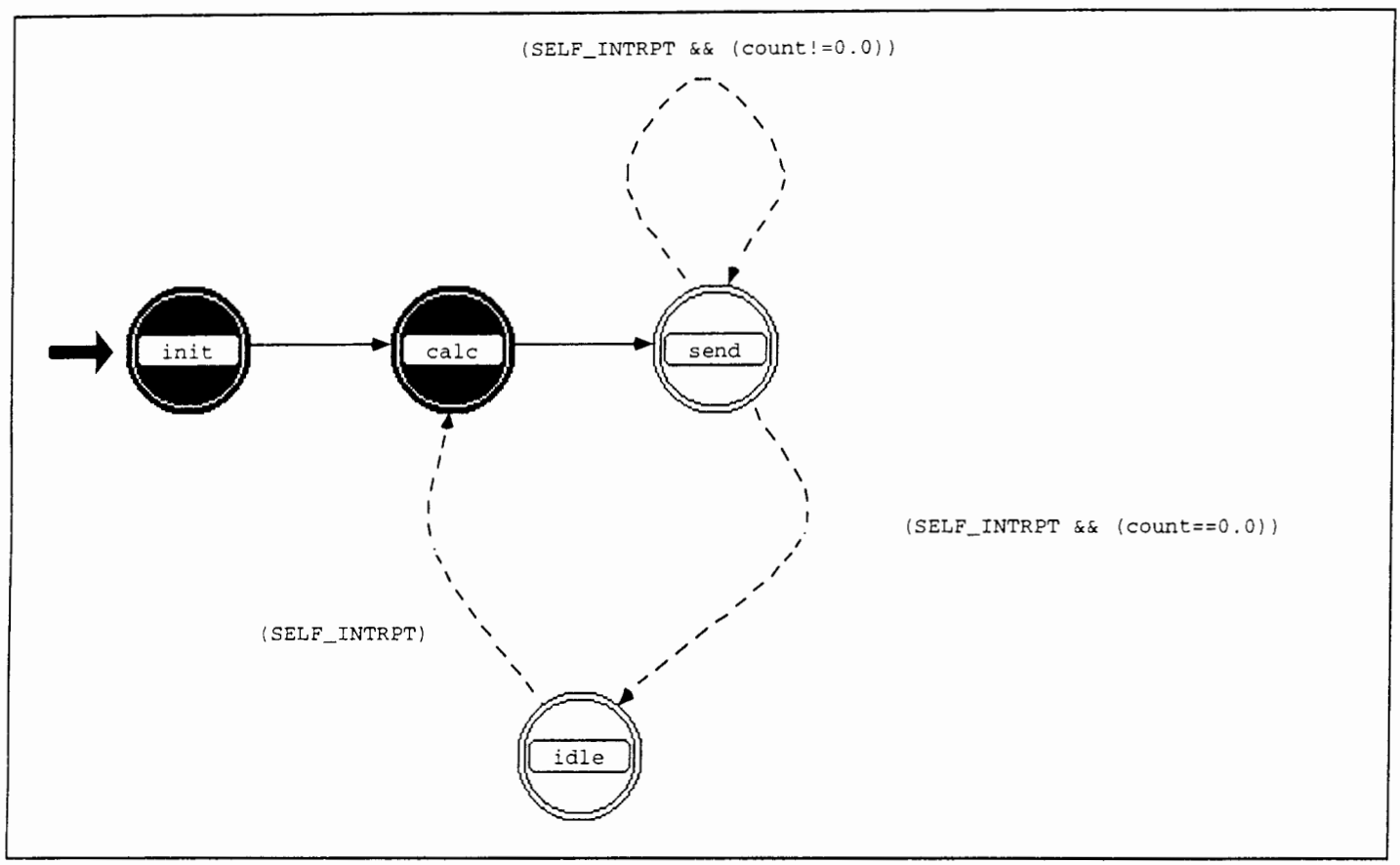
...
...

```
5   /* Create a new packet */
    pkt=op_pk_create(53);

    /* And send it */
    op_pk_send(pkt,0);
```

| *transition* **send -> send** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | tr_1 | string | tr |
| condition | default | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

# Catania VBR Source



...
...

## Header Block

```
#include "my_thesis.h"
```

## State Variable Block

```
double \count,\n;
double \inter_cell_time;
double \mean_cells;
double \mean_idle_time
Distribution *\mean_cells_dist, *\mean_idle_time_dist;
Distribution *\random_dist;
```

## Temporary Variable Block

```
Objid id;
double time;
double idle_time;
Packet *pkt;
double rand_value;
```

*forced state* **init**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | init | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

*enter execs* **init**

```
time=op_sim_time();
id=op_id_self();
op_ima_obj_attr_get(id,"mean_cells", &mean_cells);
op_ima_obj_attr_get(id,"mean_idle_time", &mean_idle_time);
op_ima_obj_attr_get(id,"inter_cell_time", &inter_cell_time);

mean_cells_dist=op_dist_load("exponential", mean_cells, mean_cells);
mean_idle_time_dist=op_dist_load("exponential", mean_idle_time, mean_idle_time);
random_dist=op_dist_load("uniform", 0, 100);
count=0;
n=0;
```

*transition* **init -> calc**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_9 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

...
...

***unforced state*** **idle**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | idle | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | unforced | toggle | unforced |

***enter execs*** **idle**

```
/*printf("In Idle\n");  */
idle_time=op_dist_outcome(mean_idle_time_dist);
time=op_sim_time();
op_intrpt_schedule_self(time+idle_time,0);
op_stat_global_write(op_stat_global_reg("Idle Time"), idle_time);
op_stat_global_write(op_stat_global_reg("cells"), n);
```

***transition*** **idle -> calc**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_8 | string | tr |
| condition | SELF_INTRPT | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***forced state*** **calc**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | calc | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

***enter execs*** **calc**

```
/*count=1;
rand_value=op_dist_outcome(random_dist);
while (rand_value>mean_cells)
{
    count++;
    rand_value=op_dist_outcome(random_dist);
}
*/
while ((count=(int)op_dist_outcome(mean_cells_dist))<0);
/*printf("Count=%4.0lf\n",count);*/
/*
op_stat_global_write(op_stat_global_reg("Number of cells"), count);
*/
```

...
...

***transition*** **calc -> send**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_4 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***unforced state*** **send**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | send | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | unforced | toggle | unforced |

***enter execs*** **send**

```
time=op_sim_time();
/*printf("In Send\n");   */
if (count!=0)
{

    /* Create a new packet */
    pkt=op_pk_create(53);

    /* And send it */
    op_pk_send(pkt,0);

  count--;
    n++;
}
```

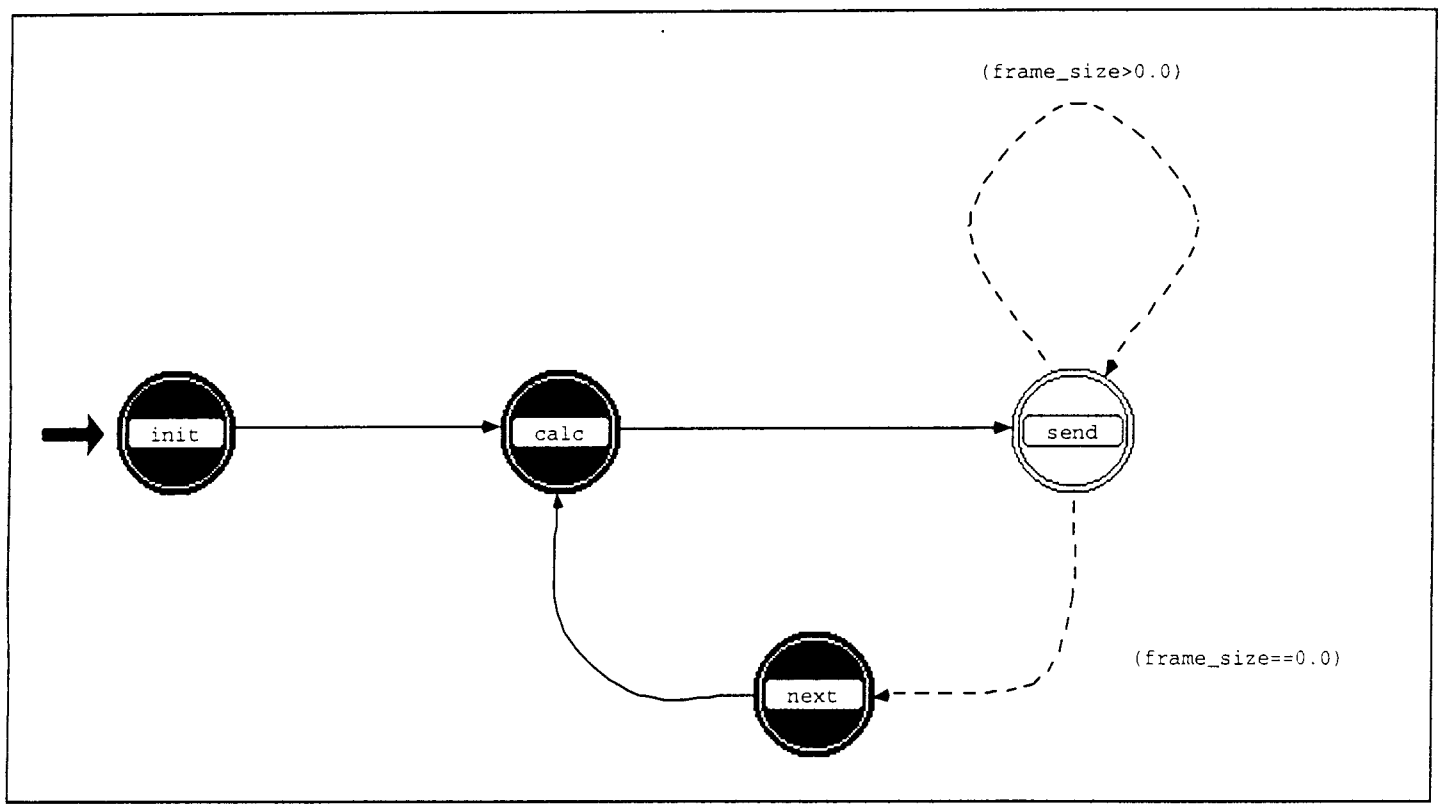
***transition*** **send -> send**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_5 | string | tr |
| condition | SELF_INTRPT && (cou... | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

***transition*** **send -> idle**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_6 | string | tr |
| condition | SELF_INTRPT && (cou... | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

# MPEG Source



...
...

| | | | |
|---|---|---|---|
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

*forced state* **calc**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | calc | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

*enter execs* **calc**

```
if (state==1)
     frame_type=I_frame;
else if ((state==4) || (state==7) || (state==10) || (state==13))
     frame_type=P_frame;
else
     frame_type=B_frame;

switch (frame_type)
{
     case I_frame:
          if (scene_size==0)
          {    while ((i_size=(int)op_dist_outcome(i_dist))<=0);
        while ((scene_size=(int)op_dist_outcome(scene_dist))<=0);
               dt=1.0/(i_size*30);
               frame_size=i_size;
               op_stat_global_write(op_stat_global_reg("Scene Size"), scene_size);

          }
          else
          {    scene_size--;
               dt=1.0/(i_size*30);
               frame_size=i_size;
          }
/*        printf("I-frame : "); */
/*        printf("Scenes left : %4.4lf\n", scene_size);  */
          break;

     case B_frame:
          while ((b_size=(int)op_dist_outcome(b_dist))<=0);
          dt=1.0/(b_size*30);
          frame_size=b_size;
/*        printf("B-frame : ");   */
          break;

     case P_frame:
     while ((p_size=(int)op_dist_outcome(p_dist))<=0);
          frame_size=p_size;
/*        printf("P-frame : "); */
          break;
}
/* printf("%4.4lf\n", frame_size);  */
op_stat_global_write(op_stat_global_reg("Frame Size"), frame_size);
op_stat_global_write(op_stat_global_reg("Mpeg Rate"), frame_size*30);
if ((frame_size*30)>max_rate)
{    max_rate=frame_size*30;
```

...
...

```
    printf("Max Rate=%lf\n", max_rate);
}
```

| *transition* **calc -> send** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | tr_1 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

| *unforced state* **send** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | send | string | st |
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | unforced | toggle | unforced |

*enter execs* **send**

```
op_intrpt_schedule_self(op_sim_time()+dt, 0);

pkt=op_pk_create(53);
op_pk_send(pkt, 0);
frame_size--;
/*printf("Frame size : %4.4lf\n", frame_size);*/
```

| *transition* **send -> send** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | tr_2 | string | tr |
| condition | frame_size>0.0 | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

| *transition* **send -> next** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | tr_3 | string | tr |
| condition | frame_size==0.0 | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

| *forced state* **next** | | | |
|---|---|---|---|
| *attribute* | *value* | *type* | *default value* |
| name | next | string | st |

...
...

| | | | |
|---|---|---|---|
| enter execs | (See below.) | textlist | (See below.) |
| exit execs | (empty) | textlist | (empty) |
| status | forced | toggle | unforced |

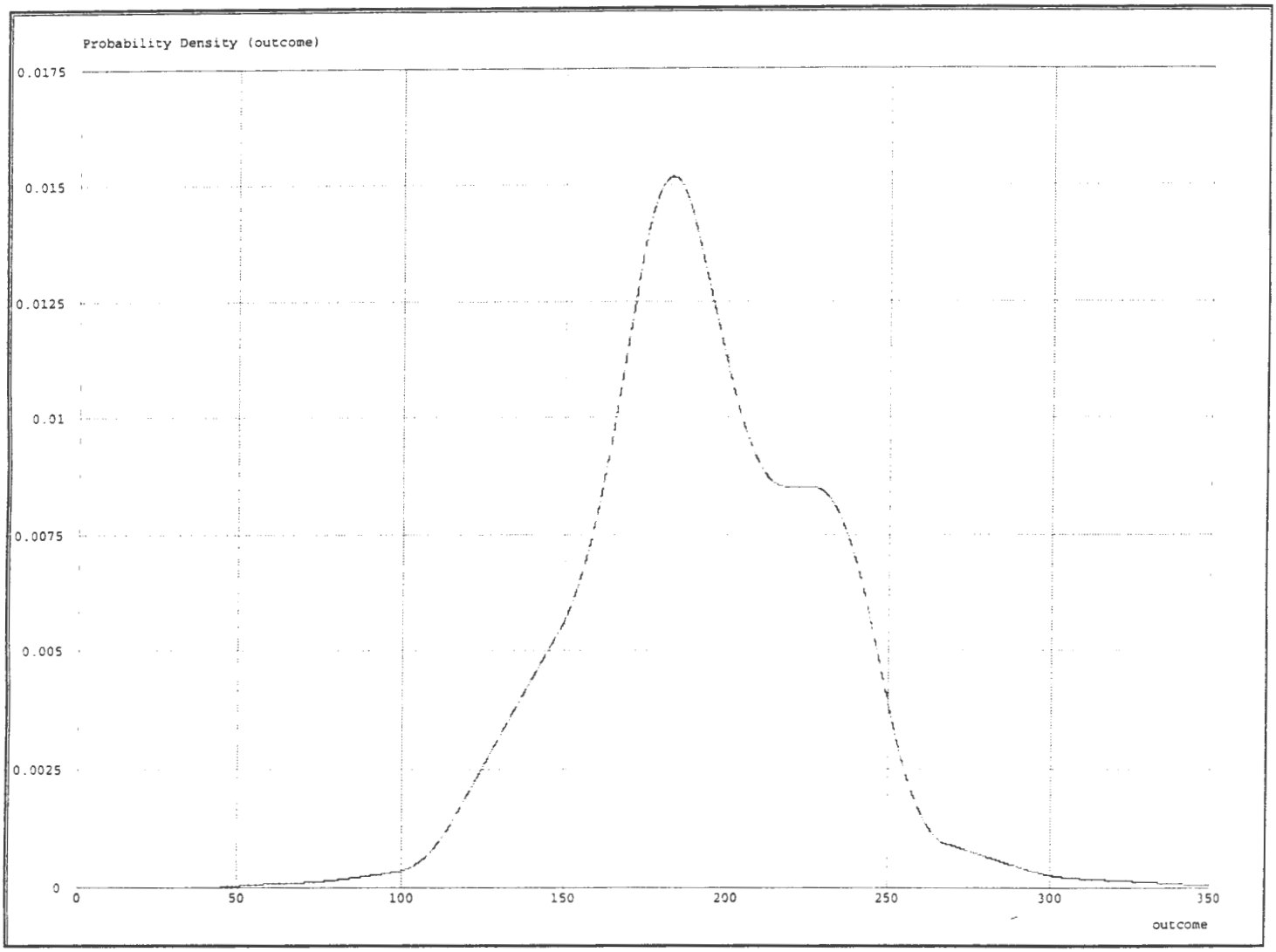
***enter execs*** **next**

```
state++;
if (state==16)
    state=1;    /* start next scene */
```

***transition*** **next -> calc**

| *attribute* | *value* | *type* | *default value* |
|---|---|---|---|
| name | tr_4 | string | tr |
| condition | | string | |
| executive | | string | |
| color | RGB333 | color | RGB333 |
| drawing style | spline | toggle | spline |

# I-frame Probability Distribution



# Scene Length Probability Distribution