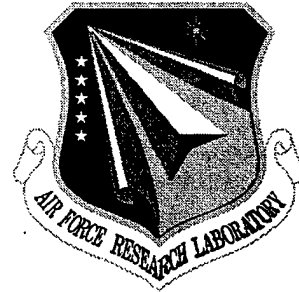


**AFRL-IF-RS-TR-1998-237**  
**Final Technical Report**  
**January 1999**



# **NEURAL NETWORKS FOR HIGH SPEED COMMUNICATION SWITCHING**

**Clarkson University**

**Robert Meyer and David Perreault**

**1 9 9 9 0 2 1 7 0 3 4**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

**AIR FORCE RESEARCH LABORATORY  
INFORMATION DIRECTORATE  
ROME RESEARCH SITE  
ROME, NEW YORK**


This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1998-237 has been reviewed and is approved for publication.

APPROVED:

*Priscilla Cassidy*  
PRISCILLA CASSIDY  
Project Engineer

FOR THE DIRECTOR:

  
WARREN H. DEBANY JR., Technical Advisor  
Information Grid Division  
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGA, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE January 1999	3. REPORT TYPE AND DATES COVERED Final Feb 94 - May 95	
4. TITLE AND SUBTITLE NEURAL NETWORKS FOR HIGH SPEED COMMUNICATION SWITCHING			5. FUNDING NUMBERS C - F30602-94-C-0023 PE - 62702F PR - 4519 TA - 22 WU - PK	
6. AUTHOR(S) Robert Meyer (Clarkson University and David Perreault (Boston University)				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Prime Contractor: Clarkson University Division of Research Clarkson Hall Potsdam NY 13699-5630 Subcontractor: Boston University College of Engineering 110 Cummington Street Boston MA 02215			8. PERFORMING ORGANIZATION REPORT NUMBER  N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGA 525 Brooks Road Rome NY 13441-4505			10. SPONSORING/MONITORING AGENCY REPORT NUMBER  AFRL-IF-RS-TR-1998-237	
11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Priscilla Cassidy/IFGA/(315) 330-1887				
12a. DISTRIBUTION AVAILABILITY STATEMENT  Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words)  Neural networks and fuzzy systems have been studied to develop effective means to control highly dynamic high-speed networks in a cost efficient manner. Using the results from a study of robust transfiguring protocols which were designed to recognize changes in assumed network state and adapt to these changing network conditions, a fuzzy control algorithm was implemented.				
14. SUBJECT TERMS  Neural Networks, Fuzzy Systems, Distributed Systems, Network Architectures			15. NUMBER OF PAGES 40	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

## Table of Contents

Abstract	ii
1. Introduction	1
1.1 Background	1
1.2 Objectives and Results	2
2. The RTNP Project	2
2.1 Protocol Performance Under Adverse Network Conditions	3
2.2 Protocol Tuning Factors	3
2.3 Identifying Network Environments	4
2.4 Protocol Control Algorithms	5
2.5 Summary of RTNP Results	5
3. Neural Networks and Fuzzy Systems	6
3.1 Architectures for Neural Networks and Fuzzy Systems	7
3.2 Microcontroller Architecture	9
3.3 Fuzzy Control Algorithms	11
4. An Example Implementation of Fuzzy Control	15
4.1 Definition of the Fuzzy Sets	15
4.2 Fuzzy Rules	17
4.3 Implementation Using a Single Microprocessor	19
5. Distributed Architectures for Neural and Fuzzy Systems	23
5.1 Future Investigations	26
Bibliography	28

## Abstract

Neural networks and fuzzy systems have been studied in conjunction with work done by SRI International in an effort to develop effective means to control highly dynamic high-speed networks in a cost efficient manner. Working with SRI, we have used the results from a study of robust transfiguring protocols. These protocols were designed to recognize changes in assumed network state and adapt protocol parameters to meet these changing network conditions. We have shown a simple microprocessor based implementation of a fuzzy control algorithm. A distributed ring architecture is proposed for distributing this algorithm in a multi-microprocessor network. Suggestions for continued development on this architecture, as well as possible future work in genetic programming are made.

## 1. Introduction

This report describes the work done for Rome Laboratory under contract number F 30602-94-C-0023. The project engineer was Mr. Charles Meyer. The work on this contract was done in close cooperation with SRI International in conjunction with their work on the Robust Transfiguring Network Protocols (RNTP) Project [1].

### 1.1 Background

Communication networks of the future will be based on high speed fiber optic links and fast digital switching. In addition, the number of nodes and links is steadily increasing. The combined effect of increased network complexity and faster transmission and switching will be to put a substantial computational burden on the processing of network control and management functions. The computations required for routing, flow control, bandwidth resource allocation, and network reconfiguration, will become the bottleneck in moving traffic through the network. Thus to fully realize the advances in optical transmission and switching technology, new computing techniques must be applied to these computationally intensive problems. Neural networks have been demonstrated as providing significant speed-ups for similar problems which can not generally be solved in reasonable time.

SRI International has begun developing new protocols for network management which are designed to adapt to changing network conditions. When networks are subjected to various kinds of stresses, protocols which were designed to perform in an optimal manner under a set of assumed operating conditions may no longer be optimal. As the network operating conditions evolve, the model on which the protocols were based is no longer valid. In their work on the RTNP project, SRI has attempted to develop methods to recognize these adverse network conditions and then to dynamically select protocols or protocol parameters in response to the observed changes in the network model.

There has been much past work on adaptive algorithms of one form or another. The difficulty with practical application of much of this work is that it relies on an accurate determination of current network state and operating conditions. Unfortunately this is seldom possible in "real-world" environments. Instead, the data is often uncertain, incomplete, or simply erroneous.

A second aspect of this problem is that because networks are inherently distributed systems, the problem solving activity must be distributed. It is well known that attempts to centralize the decision making function are

often unreliable. Centralized systems suffer from delays in getting data from remote sections of the network, or complete inability to get such data. Control decisions may be delayed or lost in reaching the intended destinations. Further, the transmission of large amounts of network status data from the entire network to a centralized decision-maker loads an already stressed network at a time when bandwidth is especially scarce. Thus, one characteristic which is highly desirable is to find methods for network management which are also distributed and do not rely on a single or a small number of decision making entities.

## **1.2 Objectives and Results**

There were two primary objectives for this work. First, we worked closely with SRI on the RTNP project to assist in defining an approach to the problem of recognizing network state under adverse conditions and in the absence of complete and accurate data from the network. Their work has been written in the form of a final report for the RTNP project [1]. In section 2 we summarize our interactions with SRI on the RTNP project and the results which we used in our subsequent work.

A second objective of our effort was to use the results from RTNP and investigate novel implementation approaches. The results from this part of the effort are described in sections 3, 4 and 5. The approach we investigated is a distributed architecture of simple microcontrollers which implement neural networks and/or fuzzy logic for decision making algorithms.

The work presented here is an initial approach based on the combined efforts of the work with SRI under the RTNP project and our own investigations. Much additional work remains to be done. It is the intention of this work to open the door to further research along these lines.

## **2. The RTNP Project**

In previous work, SRI developed the STIP3 protocol [2]. Unlike network protocols currently in use, STIP3 was designed specifically to perform multipath routing, link scheduling, and flow control in highly dynamic networks. STIP3 was designed to operate in environments with frequent changes in topology due to link jamming, node mobility, or intermittent interference. It has been shown to be superior to previously existing routing protocols based on path lengths and local queue delay.

STIP3 is a distributed algorithm which computes expected time distances to each destination node from each node. These time distances are averaged

locally and propagated to neighbor nodes only if the changes are significant (i.e. vary by more than a threshold amount from previous values). A key difference between STIP3 and most conventional routing algorithms is that STIP3 uses destination queues. Thus, each outgoing packet is placed in a queue based on its final destination. The decision as to which outgoing link should be used to route the packet toward this destination is not made until the packet is actually transmitted. This has two benefits: first, the decision is made at the last possible time instant, and thus is based on the most current network data; and second, if retransmission is required, the packet can be retransmitted over a different outgoing link. Thus a temporary or intermittent link outage, due for example to jamming, will quickly be detected and packets will be routed via alternate links as soon as possible.

## **2.1 Protocol Performance Under Adverse Network Conditions**

The first task of the work performed by SRI under the RTNP project was to investigate the performance of STIP3 under adverse network conditions. The purpose of this task was to understand which characteristics of the network operating environment would have the greatest impact on protocol performance.

So long as the operational characteristics of a network do not change radically, an advanced protocol such as STIP3 should perform well. However, under various adverse conditions, a protocol designed for superior operation may function considerably worse than its design performance. Simulations with STIP3 have shown that its performance degrades when the network is subjected to jamming (or other intermittent link outage) having various rates of jamming. A single link may be jammed continuously, in which case no packets are received at the distant end. However, if the link is subjected to a pulsed jamming, the link will successfully deliver some packets. The rate at which the link state switches between "up" (packets delivered) and "down" (no packets delivered) is the jamming rate. In networks with one or more links subjected to pulsed jamming, STIP3 does not perform as well as in those cases in which the jamming is either not pulsed or jamming rates are less dynamic.

## **2.2 Protocol Tuning Factors**

STIP3 is a complex protocol that relies on several important factors to perform well. These factors represent "tunable" parameters which must be properly selected. Another aspect of SRI's work was to determine by means of simulations which parameters could be adjusted to compensate for changes in network operating conditions.



Extensive simulations showed the following parameters to be the most effective in adapting STIP3 to changing network conditions. These were:

(a) traffic spread - a measure of the degree to which traffic is spread over multiple paths;

(b) link\_bias - a term which determines the degree to which minimum-hop routing is preferred over minimum delay routing;

(c) distance change threshold - determines if local changes in the estimated time distance to each destination are significant for propagation to neighbor nodes;

(d) time smoothing parameter - determines the degree of time averaging used to smooth estimated time distances;

(e) update period - determines the period at which update messages are sent to neighbor nodes.

These parameters are normally selected on the basis of trial evaluations of the protocol, first with simulated network operation, and then with actual network operation. Once determined, the parameters should remain valid as long as the fundamental network characteristics remain the same. However, for the purposes of this project, these parameters were identified as being the control parameters in adjusting STIP3 for varying network environments.

### **2.3 Identifying Network Environments**

The third task under RTNP was to develop techniques for recognizing and/or predicting changes in network operating environments. It was suggested that the area of artificial intelligence should be reviewed to determine if appropriate techniques were known. Based upon our suggestions and discussions with SRI, neural networks were investigated as one of two approaches. The other approach was based on previous work done by SRI in the expert systems area. This second method was based on the theory of evidential reasoning and used an in-house, SRI developed system known as GISTER. Since we had no technical input with respect to the development and use of GISTER, it will not be discussed here. Additional information is available from SRI in [3].

Neural networks were used to estimate three significant characteristics of the network environment. These were:

(a) link state probability - the probability that a given link will be "up" during the next time period;

(b) queuing delay - the estimated delay for each destination queue at each node;

(c) end-to-end jamming - a relative measure of the degree of jamming at destination nodes, averaged over all such nodes; varies between 0 (no jamming) and 1 (total jamming).

The first two of these three parameters were used directly by the STIP3 algorithm in place of the original estimates computed as part of STIP3. Simulations conducted by SRI found that the neural network estimates of both the link state probability and the queuing delay were more accurate than either the estimates produced by the original STIP3 algorithm or by the Gister-based evidential reasoning system.

The third of these parameters was not part of the original STIP3 algorithm. Instead, this parameter is used by a high-level network controller (HLNC). The HLNC was designed to adjust the tunable parameters based on changes in the network environment.

## **2.4 Protocol Control Algorithms**

The tunable parameters of the STIP3 protocol were identified in section 2.2. By controlling these parameters, the STIP3 protocol could be adapted to changes in the network environment. SRI developed a fuzzy control algorithm that uses network measurements, including the output from the neural network estimator of end-to-end jamming discussed in the previous section.

In simulations by SRI, the fuzzy control algorithm was shown to be superior to the original STIP3 algorithm for control of the "traffic spread" parameter. Recall from section 2.2, the "traffic spread" parameter controls degree to which traffic is spread over multiple links. The fuzzy rule related the value of this parameter to the estimated amount of jamming and the estimated period (1/rate) of jamming.

## **2.5 Summary of RTNP Results**

The RTNP project demonstrated the applicability of neural networks for estimating changing network states. In particular, SRI's simulations showed that neural network estimators of link state and queuing delay were significantly better than the algorithms used in the original design of the STIP3 protocol. The simulations also showed that a fuzzy control algorithm was superior to the original STIP3 protocol in its ability to vary the traffic spread as the amount of jamming and jamming rate changed.

Two limitations were noted. First, the performance of neural networks greatly depends on the training data used to establish the interconnection weights used. Thus, in designing a neural network solution, careful attention must be paid to identifying appropriate training data which reflects the expected range of operating conditions. The second limitation observed was the difficulty of discovering fuzzy rules to control protocol parameters. Although one successful rule-set was identified and demonstrated, attempts at uncovering other useful rule-sets were less fruitful. SRI has suggested that further work in this area should be done so as to develop a more complete understanding of the conditions under which fuzzy controllers would be useful.

### **3. Neural Networks and Fuzzy Systems**

Complex systems that can not be easily modeled for conventional controllers have been designed with neural network and/or fuzzy logic control. Neural networks are primarily used to implement state recognition algorithms. Fuzzy logic allows a designer to handle uncertainties in a system by conditioning the membership functions and rule base that linguistically describe the system process.

Many applications designed in industry today involve the use of microcontrollers. Some examples are cellular phones, pagers, microwaves, VCRs, electronic games, and cameras. Microcontrollers are self-contained computers in that all necessary hardware is integrated on the chip and little additional glue logic is needed. Therefore, they are ideal components to use in designing a system since the turnaround time and cost are greatly reduced. There is an abundance of software and hardware CAD tools in the market to assist system designers. These include assemblers, C-compilers, simulators, and emulators.

Algorithms already exist for fuzzy logic control on a microcontroller. Fuzzy logic code runs efficiently on microcontrollers since it requires little processing time and memory. The peripherals on a microcontroller also provide necessary interfaces for embedded control. Software tools for microcontrollers have been created that allow a designer to visualize the fuzzy control system before actual implementation takes place.

Homogeneous microcontrollers have been connected together in a network so that information can be shared among them. Microcontrollers are placed at sites in the network to process the data provided from sensors at that location. Each microcontroller handles a separate task for the network. This provides modularity to the system and eases the overall debugging process. Typically, these microcontrollers are connected in a ring or star

configuration since the number of serial communication interfaces on a microcontroller is often limited to a maximum of two. However, with a star topology, loss of the central (supervisor) controller implies failure of the network. Therefore a ring topology gives more reliability to the network and is often preferred.

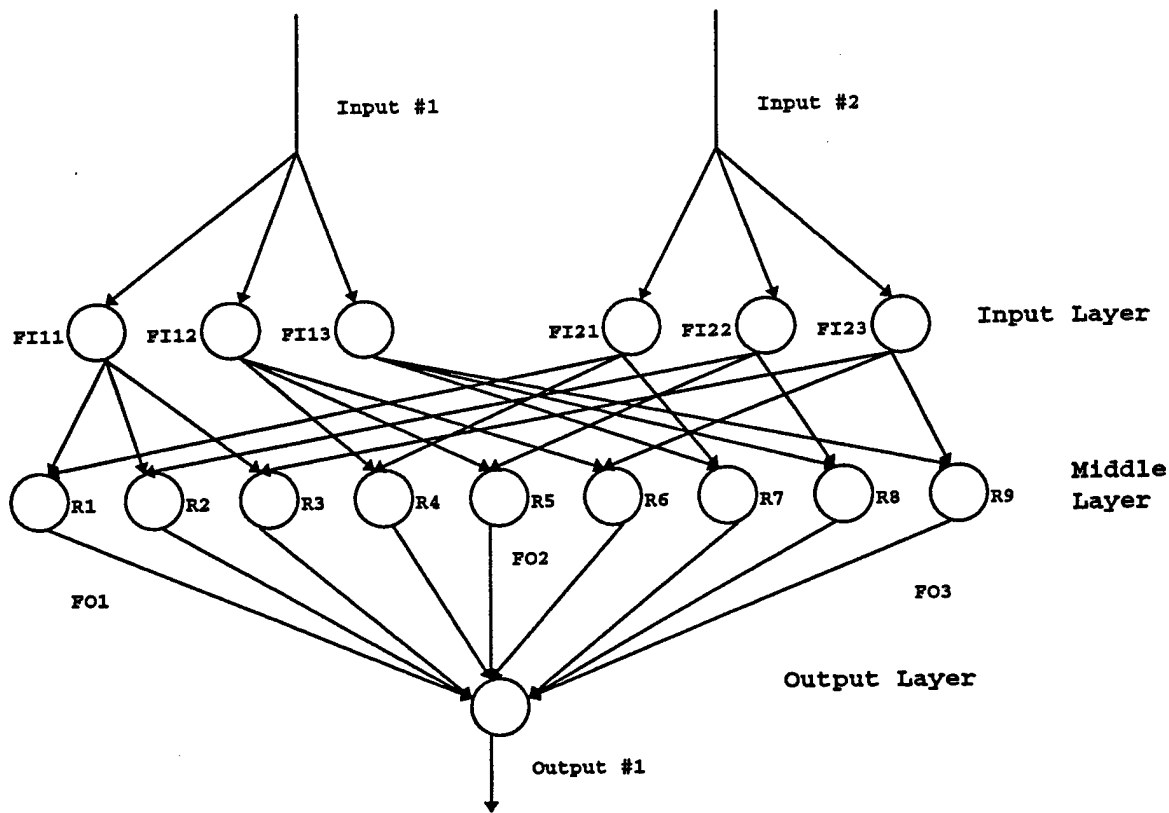
Currently, algorithms for fuzzy control are restricted to single microcontrollers. No distributed case for fuzzy logic control has been implemented as of yet. Distributing the fuzzy logic control over a ring network has several advantages. The time for completing the algorithm is shorter since the processing is overlapped at multiple nodes. There is reliability built into the system since other nodes in the network are able to process the fuzzy logic algorithm when one node fails. Inputs to the fuzzy logic control can come from sensors at remote sites. A higher number of inputs, outputs, and rules for fuzzy logic control can be handled than for a single controller.

### **3.1 Architectures for Neural Networks and Fuzzy Systems**

A methodology is needed for distributing a neural network or fuzzy logic control algorithm over a ring network given the parameters of the network and the fuzzy logic control. Architecturally there is a significant similarity between neural networks and fuzzy logic controllers. The architecture of each of these is a feedforward process, shown in Figure 1, that consists of three layers of operation. For neural networks, there is an input layer, a middle or hidden layer, and an output layer [1, 4]. For fuzzy logic, the input layer computes the fuzzification of inputs, the middle layer performs evaluation of rules, and the output layer computes the defuzzification of the outputs. There are no feedback loops in the algorithm. A static mapping of this process onto the network is not a preferable choice since the mapping is a NP-hard problem and enforces the processing at specific nodes which could fail at some time. Therefore, the distribution of the fuzzy algorithm must be dynamic in response to the resources available at that time in the network and upon the current communication delays.

The simplest distribution of fuzzy logic control in a network could be implemented by processing the algorithm only at the output nodes. This is equivalent to a static mapping of the algorithm at the output nodes. The objective of the rest of nodes in the network would be to pass the input values as message packets from the input nodes towards the output nodes. Each node in the network would maintain a queue of input packets for the fuzzy logic algorithm. Output nodes process the fuzzy logic algorithm whenever they receive the inputs that define them.

A better approach to the distribution would be to involve all nodes in processing the algorithm along the paths of communication, when it is possible. A supervisory kernel for the distributed architecture could be created that allows nodes to process packets in their queues for portions of the fuzzy logic algorithm. The kernel permits this self-scheduling if the node is idle and the time for processing the packet is less than its queue delay. Therefore there will be a dynamic response to the network resources. The nodes will be able to perform fuzzifications on the input packets, rule evaluation (if they have all fuzzy input packets that define a rule), and output defuzzification (if they have all fuzzy output packets that comprise an output). Message packets sent through the network with the kernel active could contain information on inputs, fuzzy inputs, fuzzy outputs (from rules), and outputs.



**Simple Feedforward Architecture for  
Neural Network and Fuzzy Logic Control**

**Figure 1**

Our work examines the design of a distributed network of microprocessors that implement a neural network or fuzzy control algorithm.

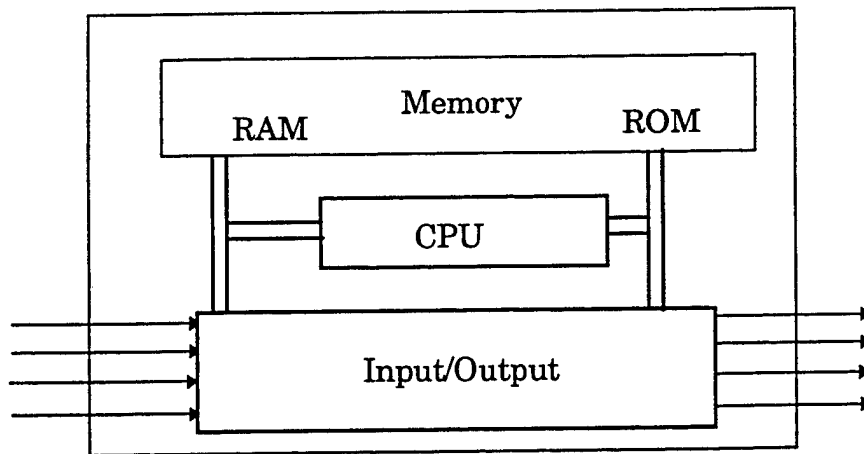
Based on the results from the RTNP project as described in section 2, both neural networks and fuzzy control algorithms were found to be superior to alternative designs for adaptive network management.

### 3.2 Microcontroller Architecture

Figure 2 displays a typical microcontroller. The main job of the microcontroller is to supervise control of the system by processing inputs and adjusting the appropriate outputs. Inputs and outputs are any digital signals; these could be from other microcontrollers, or from other digital hardware, such as digital switches in a packet switched communication network. Directing the input-output relationship is the program embedded in the controller. The number of instructions available on most CPUs is often limited to less than 100. These instructions are usually enough to manage most supervisory tasks. The strength of the microcontroller comes from its ability to be reconfigured by the instructions programmed by the user. This allows the design to be flexible in the control of the system.

The controllers that will be considered here are simple 8-bit micros. These are used because they are the most popular due to their low cost and are most familiar to most system designers. This work is intended to illustrate the basic ideas and not necessarily produce an actual prototype. At the time a prototype is designed, other choices for the actual processor should be considered in line with the technology available at that time. Several companies such as Intel, Motorola, Texas Instruments, and National Semiconductor manufacture 8-bit controllers. These chips have limited registers in the CPU and are at the low end of the performance spectrum. Yet they suffice for many typical applications. Many of the specific families derived from these controllers originated from custom designs for outside vendors. The amount of ROM typically ranges from less than 64 KBytes and the size of RAM is less than 1 KByte. These limitations can be a problem when trying to use a high-level language compiler. Different peripherals are embedded into the microcontrollers such as timers, communication interfaces, and analog-to-digital converters. Only the Motorola HC05, HC11, HC08 and the Intel 8051, 8052 8-bit micros will be mentioned following this since they are the most frequently used. The following descriptions give an idea of the basic structure of these microcontrollers.

Motorola is the leading seller of microcontrollers with the MC68HC05 device. There are over 130 variations of HC05 microcontrollers that exist, and they range in size from 16 to 44 pins on the chip. One of the smallest, the HC05K, has become so low in price that it is replacing many of the 4-bit micros in designs. The HC05 is a descendant from the 6800 microprocessor



**Typical Single Chip Microcontroller**

**Figure 2**

and the instructions for the controller are a reduced set from the 6800. The CPU has an accumulator based architecture and contains one accumulator (A), one index register (X), a program counter (PC), and a stack pointer (SP). There are 62 instructions on the chip that are used for writing programs. Ten different addressing modes are allowed for the instructions including indexing from the X register. The HC05 has a 64K addressing space and holds ROM or EPROM from 512 bytes to 32 Kbytes and RAM from 32 to 1200 bytes. There are no stack manipulation operations since the amount of RAM is so small. Up to 40 input/output pins are found on some of the chips. A 16 bit free-running timer is built into most of the HC05s for real-time interrupts and counting events. An external interrupt pin comes into the chip and is maskable from operation when necessary. Asynchronous and synchronous serial interfaces on some parts allow serial protocols in transmitting and receiving information. Various other peripherals such as A/D and D/A converters, display drivers, tone generators, and pulse width modulators (PWM) are seen on different parts. Low power modes are available to save battery life on designs that operate on batteries.

Along with the HC05, Motorola produces the HC11 and HC08 microcontrollers. Both of these chips are upward compatible from the HC05. They are higher performance parts and have more instructions and registers. Within the HC11 there are two accumulators, A and B, that help with calculations, and two 16-bit index registers IX and IY that assist in addressing. Some instructions allow 16-bit calculations by combining the A and B registers as the D register. The stack can be changed on the HC11

with push and pop instructions, unlike the HC05. Again it has a 64K internal addressing space. However some variations can be placed in an expanded mode so that it can address up to 1Meg externally. The HC11 ranges from 8 to 32K of ROM or EPROM and 256 to 1280 bytes of RAM for its memory. Additionally, the HC11 contains up to 640 bytes of EEPROM that allows the designer to have access to nonvolatile memory which can be modified during execution. The chip has 3 external interrupt lines and up to 62 possible input/output lines for interfacing. A 16-bit timer, SPI and SCI interfaces, and an A/D converter are on some parts. The HC08 is basically a faster HC05. Many instructions from the HC05 implemented on the HC08 have been streamlined for faster operation. It has only one additional register, the H register, that is called in conjunction with the X register for indexed 16-bit addressing modes. Stack operations have been included on the chip, along with a stack-indexed addressing mode. Only information on the pilot part (HC08XL36) has been released. It contains 36K ROM or EPROM and 1K of static RAM. There are 40 I/O lines on the part and 2 external interrupts. The peripherals on the chip are a 4-channel 16-bit timer, SPI and SCI serial interfaces, a phase-locked loop (PLL), and a direct memory access (DMA) coprocessor. Other parts are available with variations on the memory and peripherals.

Intel produces the 8051/8052 8-bit microcontrollers. These parts can run from frequencies of 12 to 40 MHz. One internal clock cycle for the CPU is 12 ticks of the external clock, which gives a maximum bus speed of 1 MHz. These controllers are built around a register and accumulator CPU scheme. There are banks of registers in the RAM on the chip that allow register-based instructions. The accumulator (A) is responsible for most data transfers and calculations. There are 46 instructions embedded into the 8051/52. Direct and indirect addressing modes are allowed for the instructions. These controllers have 2 to 32 Kbytes of ROM/EPROM and 64 to 256 bytes of RAM internal to the chip. They can address up to 64 Kbytes of external memory. A 16-bit pointer DPTR lets the user specify an off-chip address for data. The only difference between the 8051 and 8052 is that the 8052 has more RAM, ROM, and peripherals. There can be up to 56 I/O lines on these controllers. There are two interrupt lines, 16-bit timers, serial interfaces, watchdogs, and A/D converters available for the part. Intel has leased its design of the 8051/52 core to other companies who in turn have added other features.

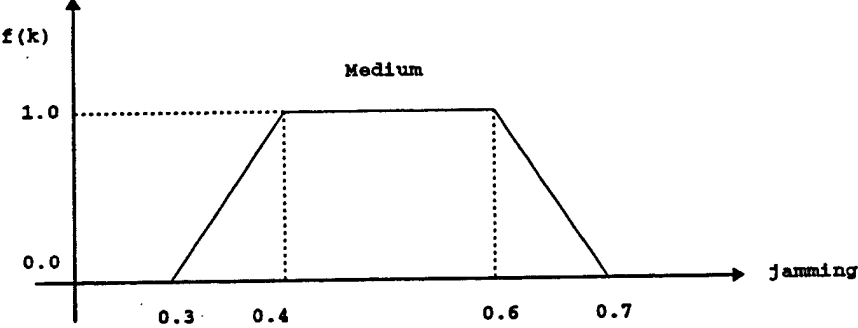
### **3.3 Fuzzy Control Algorithms**

Fuzzy logic control descends from fuzzy set theory which is an extension of traditional Boolean set theory. Membership of a set is not specifically true or false. Instead, each class of data is broken into membership regions where an element of a class can be considered partially a member of each of these



sets. Fuzzy set theory was established by Lofti Zadeh through papers [5,6] that were written on information and control in the 1960's. The advantage of fuzzy sets lies in the ability of representing data in an imprecise manner when no straightforward definition is possible. This began the development of fuzzy sets into several fields such as control and switching systems.

Some definitions of fuzzy set theory are needed. An element in the class of values  $K$  can be labeled as  $k$ . In the papers by Zadeh, a fuzzy set is said to be described by the membership function  $f_a(k)$  that translates the element  $k$  into the continuous range of (0,1). That is, the level of membership for an element  $k$  to a fuzzy set is high if the result tends toward one and low if towards zero. Unlike Boolean logic, each element can be placed into more than one fuzzy set. Therefore, an input to a class can be weighted in an approximate manner among all sets. The graph in Figure 3 gives a membership function  $f_a(k)$  for a fuzzy set of "medium" jamming. Notice that the function is subjective to the person defining it, therein comes the imprecision. The shape of the membership function can take many different forms. Besides possibly being trapezoidal as in the figure, it can be triangular, parabolic, monotonic, and bell-shaped. To see how this might be used, recall from section 2.3 that at each node, an estimate of the degree of jamming averaged over all destination nodes is computed. This estimate ranges from 0 to 1. Each node can then be classified as having the characteristic of medium jamming based on its membership function value in the fuzzy set. Some values resulting from this membership function and the data elements are shown in Table 1.



**"Medium" Jamming Membership Function**

**Figure 3**

Node	Jamming	f(k)
A	0.1	0.0
B	0.35	0.5
C	0.48	1.0
D	0.62	0.8
E	0.8	0.0

**Table 1: Fuzzy Set Values for Medium Jamming**

An empty fuzzy set means that the membership function for that set is equated to zero ( $f_a(k) = 0$ ) for all elements. Equal fuzzy sets ( $f_a(k) = f_b(k)$ ) must have identical membership functions describing them for all elements. Along with the fuzzy set definitions, there are logical operations defined for fuzzy sets. Inference rules for these operations [7] are shown in Table 2 below. These are known as "possibilistic logic" rules. Although fuzzy set values describe membership in the continuous range of (0,1), fuzzy set theory should not be confused with probability theory. Other inference rules which behave more like traditional probability rules are known as probabilistic rules.

Notation	Operation	Result
$\sim A$	not A	1-a
$\sim B$	not B	1-b
$A \cap B$	A and B	min(a,b)
$A \cup B$	A or B	max(a,b)
$A \rightarrow B$	A then B	max(1-a,b)
$A \oplus B$	A xor B	max(min(a,1-b),min(1-a,b))

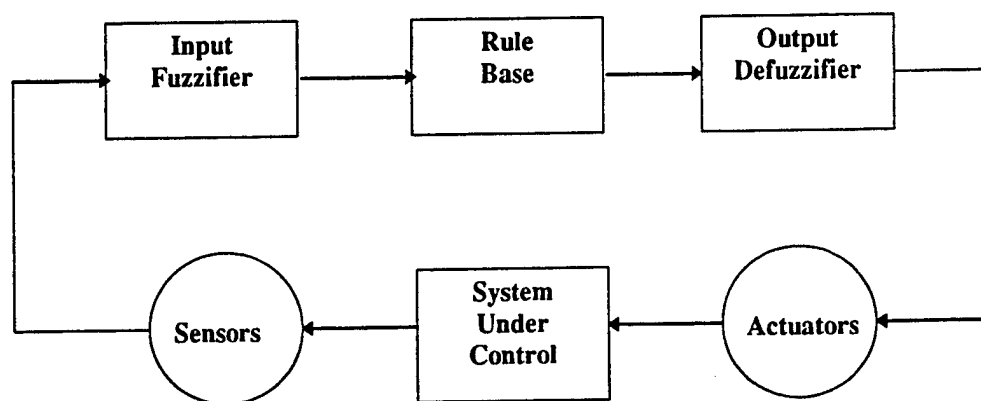
A,B = fuzzy sets    a,b = members of A,B

**Table 2: Fuzzy Set Operations**

With fuzzy logic control, the definition of membership classes allows basic rules, comprised of linguistic labels describing these classes, to define the input-output relationship of the system [8,9]. This means that heuristics and simple inference, according to the selected set of inference rules, describe the behavior of a system. It is important for the fuzzy logic control system to have defined the amount of resolution for the inputs and the discrepancy between other fuzzy sets. In other words, the number of membership functions for an input must be high enough to distinguish each of the sets and the resolution for processing the inputs must be significant enough for

good results. Without proper selection of these, it would be difficult to achieve the desired system. This also applies to the outputs of the system.

The fuzzy logic control process [10] can be broken into three major parts: fuzzifying the inputs, evaluating the rule base, and defuzzifying the outputs. Figure 4 gives the basic layout of a fuzzy logic control system. Inputs for a fuzzy logic control system can come from many various sources. For example, in RTNP, inputs were from measurements derived from protocol operation as well as neural network outputs based on recognizing network operational states. Fuzzification grades these inputs into the membership sets of the system. The rule base can be created from different methods such as human experience, process modeling, and system learning. However, most of the systems built in the area of fuzzy control today default to human experience in adjusting the rules of the knowledge base. Evaluating the rules can lead to several of them firing at one time. This could lead to a contradiction in determining the values of the outputs, if not resolved. Crisp values for the outputs come from defuzzification of their fuzzy sets. This can be calculated through many methods. One approach has been to take the center of gravity (COG) of the fuzzy outputs by their weights which is comparable to an averaging procedure. Another is the Mean of Maximum (MOM) method which examines the weight of fuzzy outputs in ratio to the number of outputs which are at their maximums. System outputs tend to be smoothed by the COG defuzzification while discrete values result from the MOM method. We have only used the center of gravity (COG) method in defuzzifying the outputs.



**Block Diagram of a Fuzzy Logic Control System**

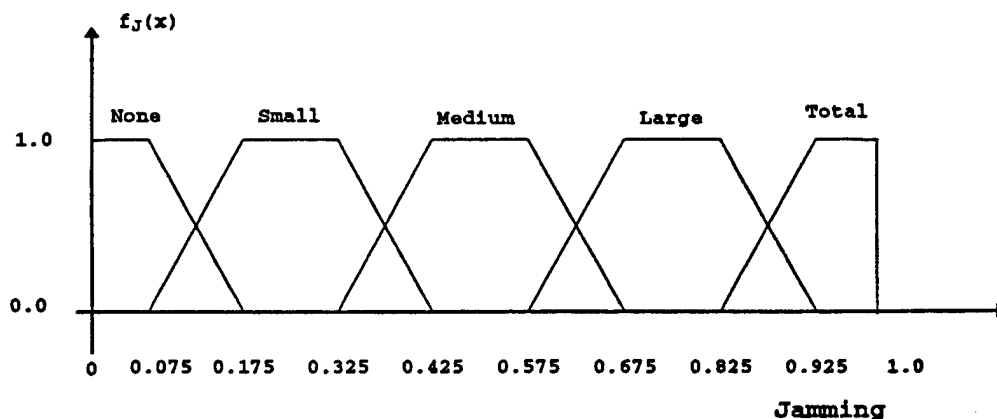
**Figure 4**

## 4. An Example Implementation of Fuzzy Control

In this section we illustrate these ideas by showing how the fuzzy control approach developed in conjunction with SRI under RTNP can be implemented in a single microcontroller subsystem. Recall from section 2.4 that the traffic spread parameter of the STIP3 protocol was determined using fuzzy logic based on the estimated degree of jamming and the estimated jamming period ( $1/\text{jamming rate}$ ).

### 4.1 Definition of the Fuzzy Sets

The first step is to relate input parameters to fuzzy values. This is done by defining fuzzy sets for jamming and jamming period. For this example, jamming is broken up into five regions: {None, Small, Medium, Large, Total} and similarly, jamming period into five regions: {Continuous, Short, Medium, Long, None}. A few words of explanation about the meaning of each of these categories is needed. For jamming, recall that this parameter is an estimate of the degree of jamming experienced by each node; it is normalized to the range of  $[0, 1]$ . Thus, if the estimate is near zero, we assign the semantics of "none" to this level. On the other hand, if the estimate is near 1, we assign the semantics of "total" to this level. For intermediate values, the fuzzy set of "small" jamming is centered at 0.25; similarly "medium" is centered at 0.5; and large is centered at 0.75. The fuzzy set membership functions are shown in Figure 5 below.



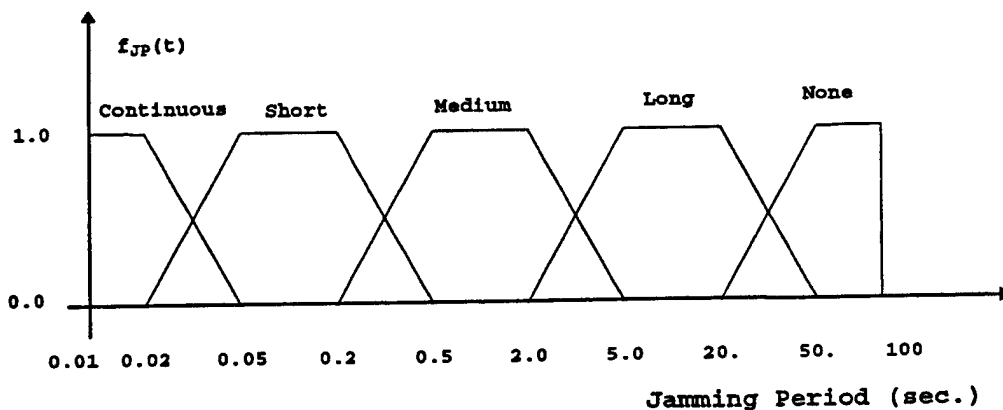
**Membership Functions for  
Jamming Input Variable**

**Figure 5**

For the jamming period, we are attempting to capture the idea of how rapidly the jamming is being applied. If the period of a pulsed jammer is very short, then we assign the term "continuous" indicating that the pulse rate is so fast as to have the same impact as if it were continuously on. As the rate slows down, the period increases and we assign appropriate terms to each of these intermediate levels. At some point, the period becomes sufficiently long that the jamming no longer has the characteristic of a pulsed jammer, but appears simply as intermittent jamming. The STIP3 protocol was designed to handle jamming of this type, and thus we assign jamming periods at this level the term "none", meaning no *pulsed* jamming. Also notice that unlike the first input, the input scale is not linear, but rather logarithmic. The logarithmic scale is used because the input covers a large dynamic range, but was judged to have only five significant levels. Observe that the fuzzy sets are centered as follows:

- (a) continuous    0.01 sec
- (b) short        0.1 sec
- (c) medium       1.0 sec
- (d) long         10.0 sec
- (e) none         100.0 sec

The membership functions are shown below in Figure 6.

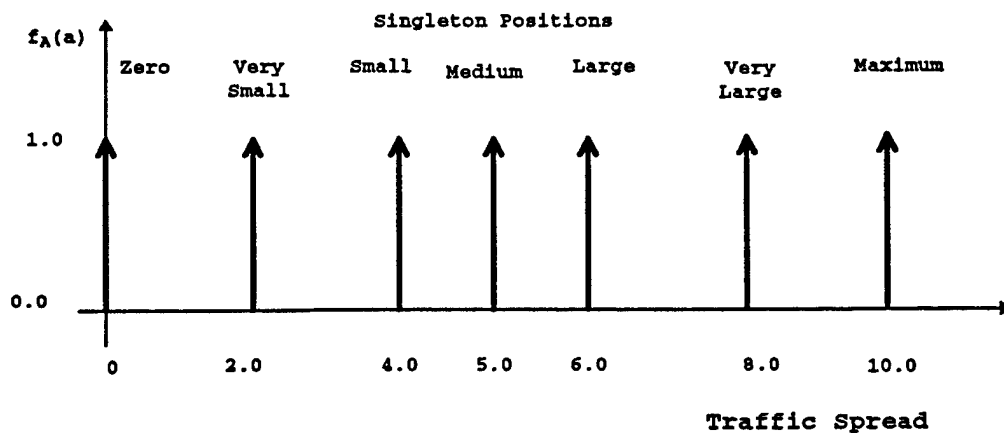


**Membership Functions for  
Jamming Period Input Variable**

**Figure 6**

In the same way that inputs are segregated into fuzzy sets, the output parameter, traffic spread is divided into 7 positions as well. The fuzzy values of the traffic spread parameter are {Zero, Very Small, Small, Medium, Large,

Very Large, Maximum}. The rules will relate fuzzy values for the two input variables to the traffic spread parameter. Several rules may be activated based on a single input combination, and therefore the collection of these fuzzy outputs must be combined. As noted in section 3, we have used the fuzzy centroid defuzzification approach to determine a single crisp output value from the fuzzy inference rules. In this case we specify the output membership functions as singleton values at the corresponding crisp value. These values for the output variable traffic spread are shown in Figure 7 below.



**Fuzzy Set Membership Functions**

**Figure 7**

## 4.2 Fuzzy Rules

The next step in determining the fuzzy control is to define the set of fuzzy rules. Each rule must capture knowledge about how the system should be controlled for each of the possible fuzzy input variable combinations. Rules typically take the form:

IF  $x$  is *fuzzy value1* AND  $y$  is *fuzzy value2* THEN  $z$  is *fuzzy output value1*

Here,  $x$  and  $y$  are input variables and  $z$  is an output variable. For the example problem,  $x$  would represent the degree of jamming, and  $y$  would represent the jamming period. We determine if " $x$  is *fuzzy value1*" by computing the membership function value for  $x$  in the fuzzy set corresponding to *fuzzy value1*. Similarly, we determine the membership

function value for  $y$  in the fuzzy set corresponding to *fuzzy value2*. The conjunction "AND" is evaluated using the possibilistic logic rules given in section 3.3, Table 2. Thus, for each rule, the minimum of the membership functions over the set of input variables for the corresponding fuzzy set values is selected. This value, in turn, determines the degree to which the output value is activated. Each rule produces a scalar value for the output variable  $z$  relative to some fuzzy set value. Since many rules may be activated, and these rules may produce scalar outputs relative to one or more fuzzy set values, some method must be used to reduce these to a single crisp value for the output variable. Several approaches have been used [4]. In this example we combine scalar values by pointwise addition over the domain of the output variable. We then compute the centroid of the resulting fuzzy set to produce a single crisp output value for traffic spread.

To illustrate, we give four possible rules below and show how these rules would be activated by various input combinations.

*Rule 1:*

*If jamming is none and jamming period is continuous, then traffic spread is very small.*

*Rule 2:*

*If jamming is small and jamming period is continuous, then traffic spread is small.*

*Rule 3:*

*If jamming is small and jamming period is none, then traffic spread is zero.*

*Rule 4:*

*If jamming is small and jamming period is medium, then traffic spread is medium*

Suppose we determine the degree of jamming to be 0.11 and the jamming period is measured as 0.03 sec. From the membership functions for jamming, we find jamming is "none" (0.65) and "small" (0.35). Similarly, jamming period is "continuous" (0.56) and "short" (0.44). This input combination activates rules 1 and 2. Rule 1 produces an output of  $\min(0.65, 0.56) = 0.56$  relative to the fuzzy set value "very small" for traffic spread. Similarly, the scalar value produced by rule 2 is  $\min(0.35, 0.56) = 0.35$  relative to the fuzzy set value "small" for traffic spread. Since no other rules are activated, we have an output which is  $0.56 * \text{"very small"}$  (2.0) and  $0.35 * \text{"small"}$  (4.0). Using the centroid calculation, we compute a crisp value as:

$$\text{traffic spread} = [(0.56 * 2.0) + (0.35 * 4.0)] / [0.56 + 0.35] = 2.77$$

Notice that in the event that only one fuzzy output set is activated, we get the "full value" of the associated crisp output value, even if the rule which activates the output is only partially activated. This can be seen in the case, for example, if jamming = 0.11, and jamming period = 4.0 sec., then only rule 4 is activated with a scalar value which is  $\min(0.35, 0.24) = 0.24$ . Since this produces an output for only the fuzzy output value of "medium", the centroid computation yields:

$$\text{traffic spread} = [0.24 * 5] / [0.24] = 5.$$

In a completely developed system with the input fuzzy sets as defined here, there could be as many as twenty-five (25) rules, one for each combination pair of input fuzzy values. In practice, it has been found [4] that often only a small fraction of the total possible number of rules are actually needed. This has two important implications. First, from a computational point of view, in a single processor environment, the computation associated with rules not actually necessary represents an additional computational load which can be eliminated. Second, in a distributed, multi-processor model, if nearly all possible rules are employed, then the system may exhibit a degree of fault tolerance if some processors fail and contribute nothing to the output calculation. This, of course, is only true to the degree that the overall performance is not particularly sensitive to a few "key" rules. For our purposes, this simple example is useful for illustrating implementation ideas. Clearly a more complete set of rules would be necessary for any practical system. Additional information with regard to the specific choices studied by SRI can be found in [1] and [11-13].

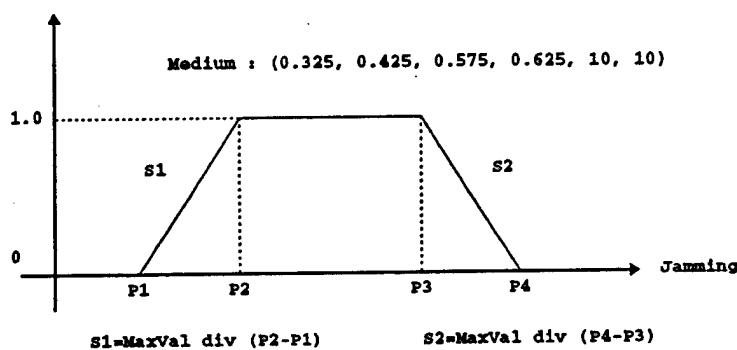
### 4.3 Implementation Using a Single Microprocessor

Microprocessors/controllers are adequate devices for performing fuzzy logic control. The ease of developing fuzzy control with microcontrollers in an embedded system is enhanced by the tools that are available for them. Three possible ways exist to compute fuzzy engine calculations: hardware, software emulation, or lookup tables. Microcontrollers typically utilize one or both of the latter two of these possibilities, and the choice of which one depends on the required speed of the control system. Operating fuzzy logic control with lookup tables is faster than software emulation in completing the algorithm. The previous example will be used to generate lookup tables on a generic microcontroller.

The membership functions, the rule base, and the singleton output values are coded into lookup tables for the fuzzy logic engine. The number of



entries for a membership function in these tables is constant so that each entry may be accessed by indexing instead of a linear search. The number of labels per input must be fixed as well for the same purpose. For the example, the membership functions are trapezoidal in shape and can be converted into a set of 6 values: {P1,P2,P3,P4,S1,S2}. This means that "Medium" jamming has a descriptor with values {0.325, 0.425, 0.575, 0.625, 10, 10} as shown in Figure 8. A similar table is created for the jamming period input. Note, however, that the log of jamming period must be computed. This is most easily done using a simple lookup table with interpolation for values not stored directly in the table. For the output variable, the singleton values are easily stored in a table indexed by fuzzy set value.



Representing a Trapezoidal Membership Function

Figure 8

To represent the rules in a general way, the number of antecedents must be allowed to be variable. Therefore the lookup table for rules must be created in a way that will allow the microcontroller to distinguish between antecedents and consequents of a rule. This may be done by marking the end of the antecedents with a specific value which is recognized as such. This could place a restriction on the number of inputs and outputs that the system can handle, but for most practical systems these limits would not be reached.

The fuzzy engine is an algorithm that first processes the inputs, then evaluates the rules, and finally determines the appropriate outputs. This algorithm is given below for a general case. Let the number of inputs be called NCI, and the number of labels (i.e. fuzzy sets) per input as LPI. If the number of cycles required to compute the membership function for one input in one fuzzy set is approximately X cycles, then the amount of time for fuzzifying all inputs is  $NCI * LPI * X$  cycles. For the rules, let the number of rules be labeled NR. The timing in processing a rule is dependent on the number of antecedents that are in that rule; however, we can take an upper bound of the total number of input variables as the number of antecedents in

each rule. If the average time of one evaluation of a rule antecedent is  $Y$  cycles, then the total time for processing the rules is  $NR*NCI*Y$  cycles. In defuzzifying the outputs, let the count of labels per output be placed as  $LPO$  and the number of outputs as  $NCO$ . Then if one output defuzzification takes  $Z$  cycles to finish and the final divide takes  $V$  cycles, the total number of cycles for all outputs is  $LPO*NCO*Z + NCO*V$ . One revolution of the fuzzy engine therefore can be said to require a total of  $(NCI*LPI*X) + (NR*NCI*Y) + (LPO*NCO*Z) + NCO*V$  cycles. Notice that the values for  $X$ ,  $Y$ ,  $Z$ , and  $V$  will depend on the type of microcontroller used to implement the fuzzy control engine. Some microcontrollers will be able to handle certain portions of fuzzy logic control better than others. The significance of this general formulation is that it shows how computational load depends on the number of input and output variables and the number of fuzzy sets used in describing these variables.

--- Fuzzify Inputs ---

```
for each input variable
{ for each fuzzy label
  { A = input variable value
    if ( A <= P1 [variable, label] or A >= P4 [variable, label])
      then member [variable, label] = 0
    else if ( A <= P2 [variable, label] )
      then member [variable, label] = (A - P1) * S1
    else if ( A <= P3 [variable, label] )
      then member [variable, label] = 1
    else member [variable, label] = (P4 - A) * S2
  }
}
```

--- Evaluate Rules ---

```
for each output variable
{ for each fuzzy label
  { fuzzyout [variable, label] = 0 }
}

for each rule
{ minrule = MAX_VALUE
  for each antecedent (an input variable and fuzzy label pair)
  { minrule = min(minrule, member [input variable, label]) }
  fuzzyout [output variable, label] =
    fuzzyout [output variable, label] + minrule
}
```

--- Defuzzify Outputs ---

```
for each output variable
{ cennum = 0
  cendenom = 0
  for each fuzzy label
  { cennum = cennum +
    fuzzyout [ variable, label] * singleton value [variable, label]
    cendenom = cendenom + fuzzyout [variable, label]
  }
  if (cendenom == 0) outputvalue [variable] = 0
  else outputvalue [variable] = cennum / cendenom
}
```

### General Algorithm for Fuzzy Controller

## 5. Distributed Architectures for Neural and Fuzzy Systems

Several steps need to be taken in order to develop a system for distributing the fuzzy logic algorithm over a microcontroller network. Code from the fuzzy logic algorithm will be placed at each processing node in the microcontroller network. This code handles only the processing of the fuzzy logic control. The kernel must supervise the packets in the queues and schedule the fuzzy logic for processing. The combination of the kernel and the fuzzy logic algorithm is used to implement the distributed fuzzy logic control. We note that although this section discusses a distributed architecture in terms of the fuzzy control algorithm, a similar approach could be used for a distributed neural network implementation. As discussed in section 3.1, the underlying architecture of a neural network system and a fuzzy controller are the same, only the details of the computational algorithm differ.

The network for distribution of the fuzzy logic control is composed of a set of homogeneous microcontrollers. These controllers are joined in a ring topology and are connected through serial communication links. The baud rate between nodes will be uniform in the network. Communication rates in the network must be much faster than the rates at which inputs arrive at the input nodes for fuzzy logic control, otherwise there will be a bottleneck at the queues of the input nodes. Processing speed at the nodes must be such that the processor is able to perform the communication task and processing task at a rate commensurate with the input data rate in order to prevent queue bottlenecks.

The fuzzy logic algorithm described in section 4 above was written for processing the algorithm serially on a single microcontroller. For the distributed case it needs to be modularized into a fuzzification unit, rule evaluation unit, and defuzzification unit. Each unit operates independently of the others, but data is passed from one to another as required by the algorithm. Each of these modules will perform one iteration of the function at a time.

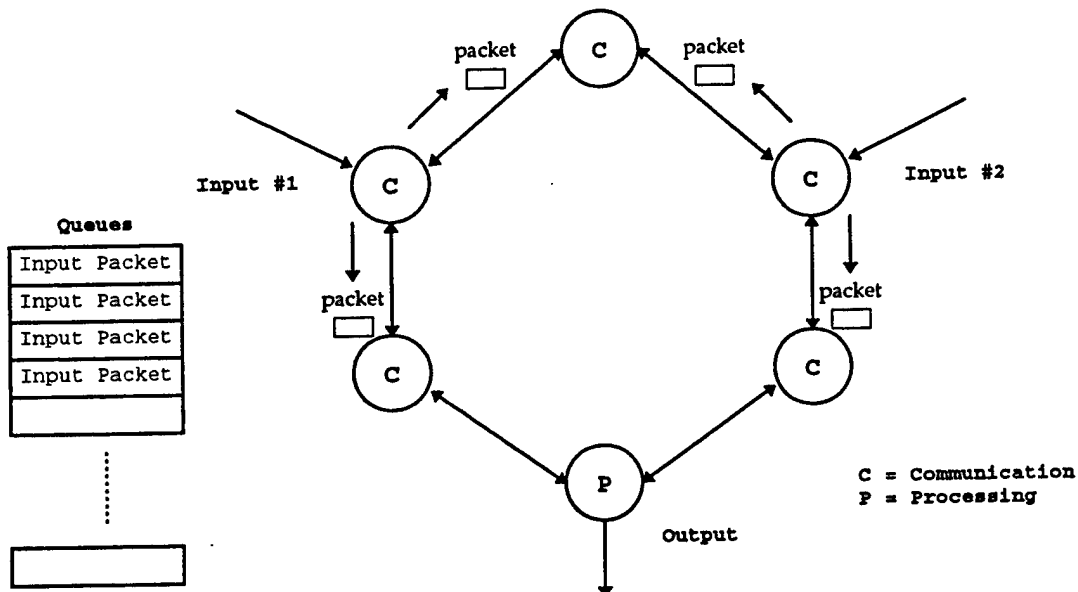
The network of microcontrollers will be exchanging data for fuzzy logic control and other processes through message packets. Message packets are restricted in size in order to minimize the communication overhead. Each of the packets for fuzzy logic control should consist of three pieces of information: an identification to indicate the type of packet (input, fuzzy input, etc...), a data block that gives the value of the packet, and a time stamp for the period when the packet was first sent. Each node will grab packets from the network depending upon the identification byte. Since a packet can be transmitted in both directions in a ring network, the time

stamp will indicate the age of the packet. A typical format is shown in Figure 9, below.



**Message Packet Layout**  
**Figure 9**

The kernel placed in each node in the network will decide which packets a node should process for the fuzzy logic control. A simple distribution scheme for the kernel would allow only the output nodes to process the fuzzy logic algorithm. When the input nodes receive the input data for the fuzzy logic control, they broadcast the information as packets towards the output nodes in both directions through the network. The output nodes would wait for the input packets that they need before processing the fuzzy logic algorithm. If an output node receives the same input packet from both directions, it ignores the later packet. Clearly the computational load is much greater on the output nodes in this case. Although this simple case could work, it does not utilize the idle time coming from the queue delays at the input nodes and does not permit a scalable design to handle increased computation. Figure 10 below illustrates this simple approach.



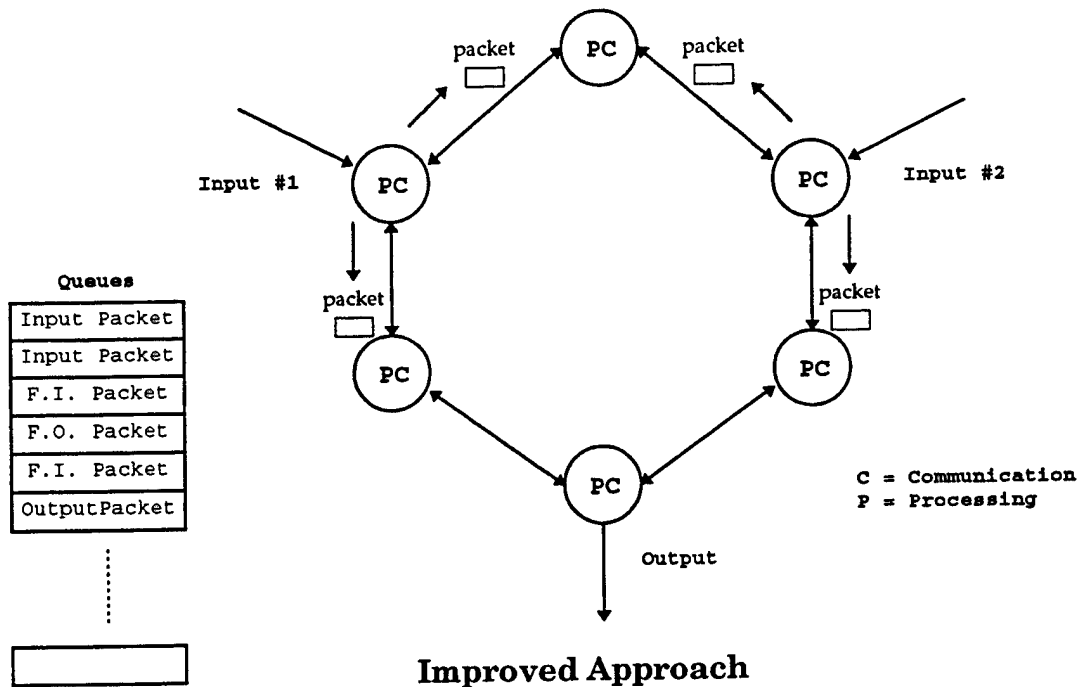
**Figure 10**

Instead of allowing only the output nodes to process the fuzzy logic algorithm, the kernel could be structured so that a node may process a portion of the fuzzy logic algorithm if:

- 1) it has the data packets necessary for it,
- 2) the amount of processing time that it takes is less than the queue delay for those packets, and
- 3) the microcontroller is idle.

This will require that the kernel sift through the queues to determine what type of information that it has at all times. Priority for processing packets in the queues will be on a first-come first-serve (FCFS) basis. If the node does perform the processing, the results are reinserted back into the queue as packets and again are broadcast towards the output nodes. Once a packet reaches the last output node in the direction that it takes through the network, it finishes its route. This method is a better approach for the distribution since the network resources are dynamically accessed dependent on the current delays of the system.

Some other considerations need to be addressed by the kernel. It is necessary to determine at the output nodes if the processing should complete whenever it receives a new input or wait until it receives all new inputs. Synchronization will not be necessary between the output nodes, since they will change their values whenever they can do so. Finally, if an output node never receives all of the information that it requires, or if there is some real-time constraint for outputs to change, there needs to be a method for degrading to some value, either abruptly or smoothly. Figure 11 illustrates this approach.



**Figure 11**

### 5.1 Future Investigations

In this section we have outlined a design approach to distribute the fuzzy logic control over a ring network of microcontrollers. A complete development of this design, testing, and verification in the context of a simulated communication network environment has not been possible within the scope of this effort. We make suggestions for the next stages of research in this area.

Examples will be needed to test the kernel for varying conditions in the network and fuzzy logic control. These examples should represent extreme cases that the kernel might encounter. Problems should be developed for multiple inputs and outputs at a single node. Also, cases should be made where the relative distance between inputs and outputs in the network is changed. The ratio of processing to communication speeds at the nodes should be varied in some examples. The number of nodes in the examples should be varied as well.

Code from the fuzzy logic engines and the kernel should be implemented in a network of microcontrollers to demonstrate the proposed approach. To verify performance, this system must be tested in a realistic network

environment. Such an environment could be provided using the simulation tools developed by SRI as part of their work on RTNP.

In addition to the distributed implementation approach described here, there have been several recent developments in fuzzy/neural systems research which should be investigated in the context of this application area. Most notably is work being done in genetic programming [14]. Some of the problems encountered by SRI in developing satisfactory algorithms for fuzzy control and neural networks might be solved using these new approaches.



## Bibliography

1. Ogier, R.G., Wong, J.S., and Khan, I.H., "Robust Transfiguring Network Protocols: Final Technical Report", ITAD-3302-FR-95-243, SRI International, Menlo Park, CA, December 1995.
2. Hight, J., Costa, E., Lee, D., Ogier, R.G., and Wong, J.S., "Evaluation and Development of Multimedia Networks in Dynamic Stress (EDMUNDS)", ITAD-8558-FR-93-277, SRI International, Menlo Park, CA, 1993.
3. Lowrance, J.D., "Evidential Reasoning with Gister-CL - A Manual", SRI International, Menlo Park, CA, 1993.
4. Kosko, B., *Neural Networks and Fuzzy Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
5. Zadeh, L.A., "Fuzzy Sets", *Information and Control*, Vol. 8, pp. 338-353, 1965
6. Zadeh, L.A., "Fuzzy Algorithms", *Information and Control*, Vol. 12, pp. 94-102, 1965.
7. Tanimoto, S.L., *The Elements of Artificial Intelligence*, Computer Science Press, New York, NY., 1990.
8. Klir, G.J. and Folger, T.A., *Fuzzy Sets, Uncertainty and Information*, Prentice Hall, Englewood Cliffs, NJ., 1988.
9. Lowen, R. and Roubens, M., *Fuzzy Logic, State of the Art*, Kluwer Academic Publishers, Boston, MA., 1993.
10. Yager, R.R. and Zadeh, L.A., *An Introduction to Fuzzy Logic Applications in Intelligent Systems*, Kluwer Academic Publishers, Boston, MA., 1992.
11. Khan, I.H., Ogier, R.G., and Wong, J.S., "Software Design Document for the Robust Transfiguring Network Protocol", SRI International, Menlo Park, CA, 1995.
12. Khan, I.H., Ogier, R.G., and Wong, J.S., "Software Test Report for the Robust Transfiguring Network Protocol", SRI International, Menlo Park, CA, 1995.

13. Wong, J.S., Ogier, R.G., and Khan, I.H., "Software User's Manual for the Robust Transfiguring Network Protocol", SRI International, Menlo Park, CA, 1995.
14. Nangsue, P. and Conry, S., "Internet-Based Genetic Programming Platform", in *Late Breaking Papers at the Genetic Programming 1997 Conference*, ed. John Koza, Stanford University, 1997.

***MISSION  
OF  
AFRL/INFORMATION DIRECTORATE (IF)***

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.