Improved Large-Set Data Transport over the Internet using Computational Intelligence Techniques

By

Jingsong Zhang

A Thesis submitted to
the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

Improved Large-Set Data Transport over The Internet using Computational Intelligence Techniques

BY

Jingsong Zhang

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Master Of Science

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank all the people who contributed towards this thesis.

I would first like to thank my advisor, Dr. Robert D. McLeod, for his guidance and encouragement through my academic years as well as his contributions and help with this thesis. In addition, I would like to express my appreciation for his kind and generous personality.

I would also like to thank Dr. Parimala Thulasiraman and Dr. Dean McNeill, for taking the time to be on my thesis committee for reading my thesis and for their efforts during my studies.

Finally, I would like to thank my wife, Jing Zhao, my daughter, Rundi, and my parents, for their constant encouragement and support.

TABLE OF CONTENTS

Chapter 1. Introduction 1

1.1 Thesis Motives 1

1.2 Structure Of Thesis 4

Chapter 2. Fuzzy Logic And Rough Sets 5

2.1 Computational Intelligence 5

2.2 Fuzzy logic 10

2.3 Rough sets 14

2.3.1 Basic Concepts 14

2.3.2 Attribute Reduction and Decision Rules 18

2.4 Summary 21

Chapter 3. Literature Review 22

3.1 Fuzzy logic in telecommunication networks 22

3.2 Summary 31

Chapter 4. Improved Data Transport Using UFTP 32

4.1 Motivation for Designing a New File Transfer Protocol 32

4.2 The UDP-based File Transfer Protocol (UFTP) 35

4.3 UFTP dynamic flow control using rough sets 40

4.3.1 Design of the rough controller 43

4.3.2 Experimental results 42

4.4 UFTP dynamic flow control using fuzzy logic 52

4.4.1 Design of the fuzzy logic controller 52

4.4.2 Experimental results 56

4.5 Summary 62

Chapter 5. Improved Data Transport Using ALRO 63

5.1 Motivation for Designing an Application Layer Routing Options (ALRO) scheme 63

5.2 Design of the ALRO scheme.....64

5.3 Experimental results 67

5.4 Summary 71

Chapter 6. Conclusions 72

References 76

LIST OF FIGURES

Figure 2.1 Processing element (neuron) 6

Figure 2.2 Three-layer feed-forward neural network 7

Figure 2.3 General structure of a fuzzy logic controller 13

Figure 2.4 Approximating the set of hired people 18

Figure 4.1 UFTP header 37

Figure 4.2 Flow chart of UFTP operations 41

Figure 4.3 Membership functions for the attributes 44

Figure 4.4 Basic Structure of a Rough Controller 46

Figure 4.5 Results of UFTP tests for different combinations of "unit/section" 48

Figure 4.6 Downloading a 15MB file with rough controller under moderate traffic scenario 49

Figure 4.7 Downloading a 30MB file with rough controller under moderate traffic scenario 49

Figure 4.8 Downloading a 100MB file with rough controller under moderate traffic scenario 50

Figure 4.9 Downloading a 15MB file with rough controller under heavy traffic scenario 50

Figure 4.10 Downloading a 30MB file with rough controller under heavy traffic scenario 51

Figure 4.11 Downloading a 100MB file with rough controller under heavy traffic scenario 51

Figure 4.12 Membership functions for *psr, cpsr*, and *fdr*, respectively 55

Figure 4.13 Transient performances of UFTPfuzzy, where *IPTD* are measured in μs 58

Figure 4.14 Downloading a 15MB file with fuzzy controller under moderate traffic scenario 59

Figure 4.15 Downloading a 30MB file with fuzzy controller under moderate traffic scenario 59

Figure 4.16 Downloading a 100MB file with fuzzy controller under moderate traffic scenario 59

Figure 4.17 Downloading a 15MB file with fuzzy controller under heavy traffic scenario 60

Figure 4.18 Downloading a 30MB File with fuzzy controller under heavy traffic scenario

..... 60

Figure 4.19 Downloading a 100MB file with fuzzy controller under heavy traffic scenario 61

Figure 5.1 Basic idea of ALRO scheme 65

Figure 5.2 Results of ALRO-UFTP, UFTPfuzzy & FTP tests 70

LIST OF TABLES

Table 2.1 An example information system 15

Table 2.2 *Hiring*: An example decision table 16

Table 2.3 *Hiring*: A reordered decision table 20

Table 2.4 *Hiring*: The discernibility matrix 20

Table 4.1 Sample decision table 44

Table 4.2 Fuzzy control rules 55

Table 5.1 ALRO vs Congestion Threshold 69

ABSTRACT

This thesis proposes and investigates two different mechanisms in order to reduce the latency perceived by Internet users when downloading large-size files from the Internet. The first mechanism is the UDP-based File Transfer Protocol (UFTP). UFTP is designed for moving large-size files over WANs where traditional file transfer protocols are found to be very inefficient. It uses UDP packets to send data to avoid round trip propagation delay, a limiting factor in data transferring. A dynamic algorithm based on fuzzy logic or rough sets is designed for UFTP flow control. Experimental results presented show that UFTP, with flow control using fuzzy logic or rough sets, is up to 1.63 times faster than the conventional FTP.

The second mechanism is the Application Layer Routing Options (ALRO). Most file transfer architectures are based on the conventional client server paradigm with the Internet providing the interconnect infrastructure. In order to avoid/control network traffic congestion, the most common scenario is that the server effectively throttles back its transmission when congestion is inferred. But with the new mechanism ALRO, the server will not throttle back its transmission when congestion is inferred; instead, it will reroute the flow of packets along an alternative route constructed through a relay router or server. The proposed mechanism is implemented through UFTP. Experimental results presented show that ALRO (combined with UFTP) works as expected: attempting to balance the network load when congestion is inferred while still obtaining high–speed file transferring.

Chapter 1

Introduction

1.1 Thesis Motives

With the rapid development of the Internet over the past few years, more and more Internet users tend to complain about the time that they waste sitting behind their computers, doing nothing but waiting for data to download. This concern has grown larger with the emergence of multimedia files over the Internet. With more and more large files of several hundred megabytes on today's Internet, latency has become a big concern for a typical Internet user. Minimizing the latency perceived by Internet users has become an important performance objective.

Latency, which simply means delay, can be caused by a number of sources: client and server hardware, the network itself, and Internet protocols. If a server is overloaded or has a slow disk, or if a user's computer is not quick enough to parse the incoming packets and process them, it imposes a considerable delay as perceived by the user. Latency like this caused by a server or a client can be largely eliminated simply using more powerful computers, more memory, or faster disks. Networks can also cause latency. Some sources of this delay are intrinsic to the network infrastructure, namely transmission delays. Since most Internet users nowadays have access to high-speed Internet connections, transmission delay is not a big concern any more. Network congestion can also increase

latency, but this problem can be alleviated to a great extent with high performance routing algorithms and a fast and reliable network infrastructure.

However, remedies to client and server hardware as well as the network do not compensate for inefficient protocols. The design and implementation of inefficient Internet protocols is another cause of delay; for instance, transport protocols were initially designed to match particular network characteristics with the type or size of the data to be transmitted. With the evolving nature of the Internet, these protocols are constantly modified in order to optimize performance, even though these modifications are often limited or constrained by legacy implementations.

Many modifications to the existing protocols have been proposed in the literature to reduce latency; some of them have been already implemented on the Internet. The most significant modifications include: avoiding the cost of round trip time by reducing the number of Hyper Text Transport Protocol (HTTP) connections [1], using multiple Transmission Control Protocol (TCP) connections to the server, currently used by web browsers that comply with HTTP1.1 [2], using multiple TCP connections in conjunction with the File Transfer Protocol (FTP) [3], better estimating bandwidth delay products in adjusting TCP parameters [4], using pre-fetching and caching techniques to reduce Web latency [5-16], and using mirror/replicated servers to handle the HTTP requests [17-20].

FTP and HTTP files contribute the largest volume of traffic on today's Internet. FTP and HTTP both are implemented on top of TCP. TCP was originally designed in late 1970's

for the first version of the Internet, the ARPA Network (ARPANET), with only one task in mind, to "provide reliable transfer of stream information over the connectionless Internet Protocol (IP)"[21, p28]. Its slow-start algorithm for congestion control and "lock step" sliding-window mechanism for flow control and the retransmission of each lost packet has become a bottleneck on today's high-speed and reliable Internet [22-24]. One way to solve this problem is to modify TCP, another way is to avoid TCP, e.g., use the User Datagram Protocol (UDP) and implement newer and much better congestion or flow control mechanisms at higher layers within the protocol stack.

In this thesis, we investigate ways to improve large-set data transport over the Internet. Computational Intelligence (CI) techniques using fuzzy logic and rough sets are investigated and demonstrated for use in solving flow control problems.

CI techniques are a set of heuristics that exhibit characteristics of intelligence, such as learning, flexibility and adaptation. It is advocated that such characteristics are necessary for the effective support of complex systems. It is observed that there is a trend in today's networking community: integrating all services over the same network infrastructure. This trend introduces new, more complex problems, whose solutions are well within the application domain of CI techniques.

Finding quicker and more efficient ways integrated with computational intelligence techniques to transfer large-size files over the Internet and thus reducing the latency perceived by an Internet user is the major motivation for this thesis. Two different

mechanisms will be proposed and investigated in this thesis. The first mechanism is the UDP-based File Transfer Protocol (UFTP) [25] which attempts to improve large-size file transport across a large WAN. A dynamic algorithm based on fuzzy logic or rough sets is designed for UFTP flow control. The second mechanism is the Application Layer Routing Options (ALRO) [26] whose objective is to investigate ways in which the network and its topology itself can be more efficiently utilized when dealing with large-set data transport over the Internet.

1.2 Structure of Thesis

The remainder of this thesis is organized as follows. In Chapter 2, an introduction to fuzzy logic and rough set theory is given. A literature review is provided in Chapter 3. In Chapter 4, a new UDP-based file transfer protocol (UFTP) is introduced and explained in detail. The design of the flow controllers for UFTP using rough sets and fuzzy logic is also explained in this chapter. Chapter 5 provides in detail the design of the Application Layer Routing Options (ALRO) scheme. The conclusion as well as some future research directions is presented in Chapter 6.

Chapter 2

Fuzzy Logic And Rough Sets

2.1 Computational Intelligence

Computational Intelligence has emerged as an important field in computing. It includes an emerging and more or less established family of problem-stating and problem-solving methods that attempt to mimic intelligence. Neural networks, evolutionary computing, fuzzy logic and rough sets constitute the backbone of the domain of Computational Intelligence. In this section, we identify the key features for neural networks and evolutionary computing. Fuzzy logic and rough sets will be described in the following two sections.

Neural networks are distributed computing units that are equipped with significant learning abilities. Starting from pioneering research of Rosenblatt [27] and Minskey [28], neural networks have undergone a significant change, and become an important supply of learning methods.

Formally, a neural network is a parallel, distributed, information processing structure consisting of many processing elements interconnected via weighted connections [29]. A processing element in a neural network is called a *neuron*, and its generic structure is shown in Figure 2.1.

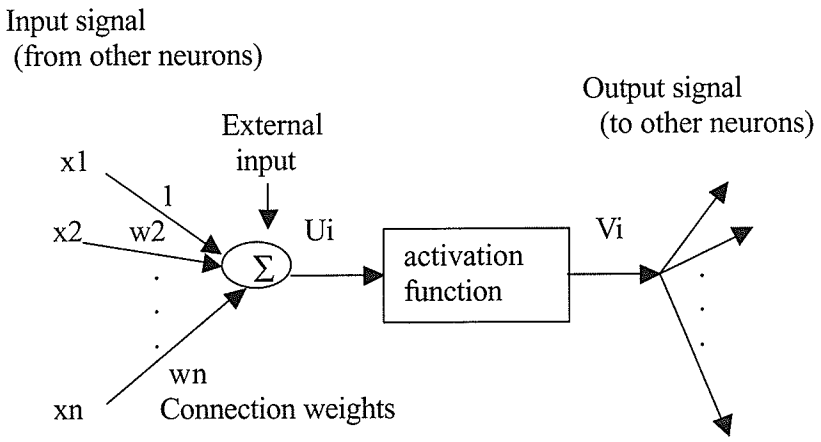


Figure 2.1. Processing element (neuron)

For the neuron shown in Figure 2.1, the net input U_i is

$$U_i = \sum_{j=1}^{n} w_j x_j + I_i,$$

where I_i is the external input. The relationship between the net input U_i and the net output V_i is characterized by an activation (transfer) function, which is typically a nonlinear, continuous, and differentiable function. The following activation function, called *sigmoid function*, is widely used:

$$V_i = \frac{1}{1 + \exp(-gU_i)},$$

where g is a gain parameter. The sigmoid output V_i ranges from 0 to 1. A neural network consists of large numbers of these simple neurons that are connected to each other and operate in parallel. The fundamental problem with the neural network is how to determine the connection topology and the connection weights between neurons.

There are two popular models of neural networks: the *feed-forward model* and the *feedback model*. In the feed-forward model, the neurons are arranged in layers, as shown in Figure 2.2.

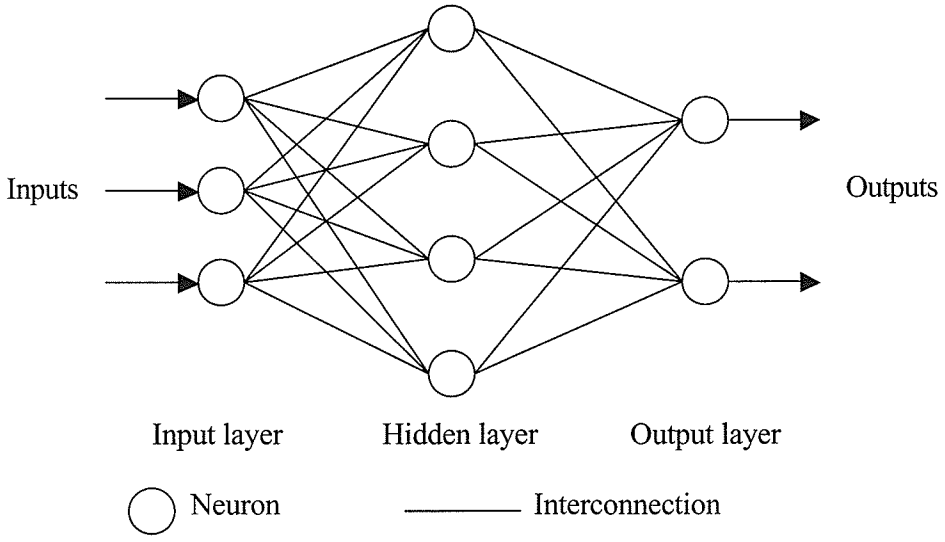


Figure 2.2. Three-layer feed-forward neural network

This neural network model is often called *multi-layer perceptron*. The inputs are applied to the input layer, and the outputs are collected from the output layer. A three-layered feed-forward neural network consisting of an input layer, a hidden layer, and an output layer, has been widely studied. It can map an arbitrary nonlinear function by adjusting connections weights through the learning (or training) process. This type of neural network has been used for nonlinear control, adaptive control, and pattern classification. For the feed-forward neural networks, a training algorithm called the *back-propagation learning algorithm* is widely used [29]. In the training phase, the neural network is given

a set of input values and the matching desired output values, and the connection weights are adjusted according to the training data. All the weights are modified by backward error corrections according to the following learning equation:

$$W^{new} = W^{old} - \alpha \frac{\partial (Y(X) - F(X))^2}{\partial W},$$

where W, X, $Y(X)$ and α represent the connection weights, inputs, desired outputs, and learning rate, respectively. $F(X)$ represents the neural network output values for the inputs X. This equation allows the connection weights to adjust so that the mean squared error between the neural network output and the desired output is minimized. After being trained, the neural network generates the correct output values for a given set of input values. Even if the inputs are outside the training data set, the neural network gives the answers that are generalized and best fitted for the inputs.

In the feedback neural network model, one or more feedback loops exist in the system, i.e., the outputs of one or more elements in the system influence in part the inputs applied to those particular elements. The feedback in a feedback neural network can be of local or global kind. Recurrent networks are examples of such kind of neural networks. One of the most prevalent uses of this type of neural network is to solve constrained optimization problems by casting an optimization problem into the form of an energy function that describes the dynamics of a neural system. Due to the massive parallelism and possibly fast convergence to solutions, neural networks can eliminate the time bottlenecks that usually arise in most traditional sequential algorithms.

Neural networks are universal approximators: they can approximate any continuous function to any required degree of accuracy. Thus, neural networks are capable of representing any nonlinear mapping to the required accuracy. The approximation is realized through a learning process in which the connections are updated to follow the required target values of the mapping.

Evolutionary Computing gleans concepts from 'natural evolution' to perform optimization, search and synthesis of information. According to the nature of the algorithms, there are several categories of Evolutionary Computing, such as, evolutionary strategies, genetic algorithms, and evolutionary programming. Rechenberg [30] introduced 'evolutionary strategies' as an optimization technique. He started with a 'population' of two individuals, one parent and one offspring. The offspring was produced by a random change (mutation) in the parent. Genetic algorithms, which are based on the Darwinian model of survival of the fittest, were invented by John Holland [31], where he used a number of genetic operators, like 'crossover', 'mutation', and 'inversion'. Fogel, Owens, and Walsh [32] developed 'evolutionary programming', an optimization technique where candidate solutions were represented as finite-state machines. Among these three techniques genetic algorithms have recently become very popular due to Goldberg [33]. They have been successfully applied in many areas such as machine learning, image processing, manufacturing, etc.

2.2 Fuzzy Logic

Fuzzy logic, also termed granular computing, is one of the main areas in Computational Intelligence. It is based on fuzzy set theory initiated by Zadeh in 1965 [34]. It arose from the desire to emulate human thought processes that are imprecise, deliberate, uncertain, and usually expressed in linguistic terms. The technique allows us to model a system using 'vague' or 'ambiguous' terms (as we often use in real life). It also has the advantage of establishing an interface between symbolic and numerical data, i.e., it has the capacity to exploit imprecise data expressed in natural language by human observers or experts, at the same time with precise numerical data provided by measurement devices.

Fuzzy logic control was introduced by Mamdani [35] for the control of complex processes, such as industrial plants, especially when no precise model of the processes exists and when most of the priori information is available only in qualitative form. It has been observed that a human operator is sometimes more efficient than an automatic controller in dealing with such systems. The intuitive control strategies used by trained operators may be viewed as fuzzy algorithms, which provide a possible method for handling qualitative information in a rigorous way.

The basic idea of fuzzy logic control is to make use of expert knowledge and experience to build a rule base with linguistic rules. Proper control actions are then derived from the rule base, which can be considered as an emulation of the human operator behavior. Different from other control methods, fuzzy logic control does not involve complex

mathematical operations and models of systems. Its design theory does not explicitly suggest a solution for a particular control problem. The question of how to solve a control problem in fuzzy control is assumed to be the responsibilities of the experts. Consequently, the design of a fuzzy logic controller depends entirely on the knowledge and experience of the experts.

The basic principle used in fuzzy logic control is the notion of a fuzzy linguistic rule. A fuzzy rule is a conditional statement, typically expressed in the form "if-then". As an example, one of the rules in our fuzzy flow controller (see Section 4.4) is: "If the packet success rate is *moderate* and the change of packet success rate is *decreasing_fast*, then the fractional delay rate should be *very_high*". In this relational expression "packet success rate", "change of packet success rate" and "fractional delay rate" are called *linguistic variables* which accept values among the words of a natural or synthetic language such as "moderate", "decreasing_fast" and "very_high". Possible values of the linguistic variables are called *linguistic values* and they are modeled by fuzzy sets [34]. The deduction of the rule is called the inference and requires the definition of a *membership function* characterizing this inference. This function allows us to determine the degree of truth of each proposition. Various implication methods have been developed to implement the inference mechanism.

Figure 2.3 shows the general structure of a fuzzy logic controller. The basic functions of components in the fuzzy logic controller are described as follows [36].

- *Fuzzifier*: The fuzzifier converts the input system performance parameters into suitable linguistic values which are needed in the inference engine.
- *Fuzzy Rule Base*: The fuzzy rule base contains a set of fuzzy control rules, defined in a linguistic way, to describe the control policy.
- *Inference Engine*: The inference engine infers the fuzzy control action under the fuzzy control rules and the related input linguistic parameters.
- *Defuzzifier*: The defuzzifier converts the inferred fuzzy control action into a non-fuzzy control action under a defuzzification strategy.

To summarize, a fuzzy inference engine maps fuzzy sets to fuzzy sets. In control engineering applications we almost always deal with numerical values. The *Fuzzifier* and *Defuzzifier* modules act as interfaces between linguistic world of the fuzzy inference engine and numerical world. The *Fuzzifier* module takes a numerical value and maps it to a fuzzy set, while the *Defuzzifier* module takes a fuzzy set and produces a non-fuzzy output whose objective is to represent the possibility distribution of the inference.

Complete knowledge of the controlled system is not required to design a fuzzy logic controller. However, it does require a good expertise and experience with the process in order to build the fuzzy rules that are the basis of the controller. Moreover, the definition of the membership functions is crucial. The "distance" between the peaks of two membership functions is especially critical. Too small a distance will lead to oscillations while too big a distance will lead to a large steady-state error.

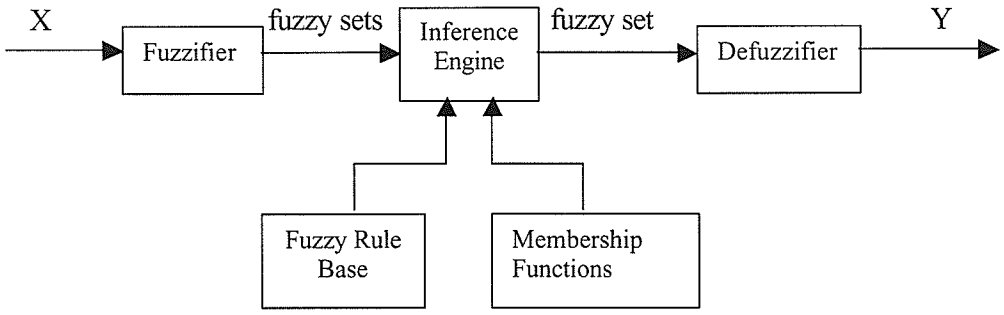


Figure 2.3. General structure of a fuzzy logic controller

There are two important principles supported by fuzzy logic and fuzzy sets [37]. The first, formulated by the founder of fuzzy sets, Zadah, is called the principle of incompatibility [38]. In brief, this principle promotes fuzzy logic and fuzzy sets as a basic vehicle useful in overcoming an evident and acute disparity between precision and relevancy when modeling complex phenomena. The second principle originating within the realm of computer vision [39] alludes to an important idea of least commitment and graceful degradation that advises to postpone any final decision until the point where enough evidence has been gathered. In this sense, fuzzy logic and fuzzy sets allow us to quantify uncertainty and take advantage of it rather than blindly discard it.

Usually, fuzzy logic and fuzzy sets, as well as rough sets, promote top-down design approaches. In particular, we first capture the skeleton of the problem and subsequently refine the solution by moving into processing granules. When dealing with fuzzy sets, all processing is carried out at the level of such information granules. The numeric details are hidden on purpose.

2.3 Rough Sets

Rough sets have been introduced as a tool to deal with inexact, uncertain or vague knowledge in computational intelligence applications. It provides a means of deriving rules from decision (data) tables. The derived rules are then used as a basis for evaluating as well as controlling processes. In some ways, rough set theory is a complementary notion to fuzzy sets, but the concept of rough sets is fundamentally different from the idea of fuzzy sets. Fuzzy sets allow partial set memberships to handle vagueness, while rough sets allow multiple set memberships to deal with indiscernibility.

2.3.1 Basic Concepts

Rough set theory is based on a mathematical model for information systems and derivation of rules from what are known as decision tables [40-42]. For computational reasons, a syntactic representation of knowledge is provided by rough sets in the form of *information systems*, or *data tables*. Informally, a data table is represented as a collection of rows each labeled with some form of input (a case, an event, a patient, or simply an object), and each column represents an attribute (a variable, an observation, a property, etc.) that can be measured or computed for each row input; it can also be supplied by a human expert or user. More formally, a data (information) table is represented by a pair (U, A), where U is a non-empty, finite set of objects and A is a non-empty, finite set of attributes, where $a: U \rightarrow V_a$ for every $a \in A$. The set V_a is called the value set of a.

Table 2.1 shows a very simple information system with nine cases or objects, and four attributes (*Diploma*, *Experience*, *French* and *Reference*). Notice that the cases *x1* and *x2*

have exactly the same values of conditions. The cases are (pairwise) indiscernible using the available attributes.

In many applications, there is an outcome of classification that is known. This posteriori knowledge is expressed by one distinguished attribute called the *decision attribute*; the process is known as supervised learning. Information systems of this kind are called *decision tables*. Formally, a decision table is any information system of the form $(U, A \cup \{d\})$, where $d \notin A$ is the decision attribute. The elements of A are called *conditional*

	Diploma	*Experience*	*French*	*Reference*
x1	MBA	Medium	Yes	Excellent
x2	MBA	Medium	Yes	Excellent
x3	MBA	Low	Yes	Neutral
x4	PhD	Low	Yes	Good
x5	MSc	High	Yes	Neutral
x6	MSc	Medium	Yes	Neutral
x7	MSc	High	Yes	Excellent
x8	MBA	High	No	Good
x9	PhD	Low	No	Excellent

Table 2.1 An example information system.

attributes or simply *conditions*. The decision attribute may take many values, though binary outcomes are rather frequent. Table 2.2 shows a small example decision table. The table has the same nine cases as in Table 2.1, but one decision attribute *Decision* with two possible outcomes {Accept, Reject} has been added. Notice that the indiscernible cases *x1* and *x2* have different outcomes. One possible reason for this may be that the company didn't have so many openings for every qualified person.

	Diploma	Experience	French	Reference	Decision
x1	MBA	Medium	Yes	Excellent	Accept
x2	MBA	Medium	Yes	Excellent	Reject
x3	MBA	Low	Yes	Neutral	Reject
x4	PhD	Low	Yes	Good	Reject
x5	MSc	High	Yes	Neutral	Accept
x6	MSc	Medium	Yes	Neutral	Reject
x7	MSc	High	Yes	Excellent	Accept
x8	MBA	High	No	Good	Accept
x9	PhD	Low	No	Excellent	Reject

Table 2.2 *Hiring*: An example decision table.

A decision table expresses all the knowledge about the model. This table may be unnecessarily large. For example, the same or indiscernible objects may be represented several times, or some of the attributes may be superfluous. We are going to look into the first issue now. The second issue will be discussed in the next section.

Let (U, A) be an information system, then for each $B \subseteq A$, there is associated an equivalence relation $\mathrm{Ind}_A(B)$, called *B-indiscernibility relation*, such that

$$\mathrm{Ind}_A(B) = \{(x, x') \in U^2 \mid \forall a \in B.\, a(x) = a(x')\}$$

If $(x, x') \in \mathrm{Ind}_A(B)$, then objects x and x' are indiscernible from each other relative to attributes from B. This is a fundamental concept in rough sets. The notation $[x]_B$ denotes equivalence classes of $\mathrm{Ind}_A(B)$. The relation $\mathrm{Ind}_A(B)$ defines a partition of the universe U. If we consider, for instance, $B = \{Diploma\}$ with the data table shown in Table 2.1, objects *x4* and *x9* belong to the same equivalence class and are indiscernible. It's also easy to see that

$$\mathrm{Ind}_A(\{Diploma\}) = \{\{x1, x2, x3, x8\}, \{x4, x9\}, \{x5, x6, x7\}\}$$

Let $B \subseteq A$ and $X \subseteq U$, then the set X can be approximated from information only contained in B by constructing a B-lower and B-upper approximation denoted by $\underline{B}X$ and $\overline{B}X$ respectively, where $\underline{B}X = \{ x \mid [x]_B \subseteq X \}$ and $\overline{B}X = \{x \mid [x]_B \cap X \neq \varnothing\}$. A lower approximation $\underline{B}X$ of a set X is actually a collection of objects that can be classified with full certainty as members of X using the knowledge represented by attributes in B. By contrast, an upper approximation $\overline{B}X$ of a set X is a collection of objects representing both certain and possible uncertain knowledge. The set $BN_B(X) = \overline{B}X - \underline{B}X$ is called the *B-boundary region* of X, and thus consists of those objects that we cannot decisively classify into X on the basis of knowledge in B. The set $U - \overline{B}X$ is called the *B-outside region* of X, and thus consists of those objects that can be with certainty classified as do not belonging to X (on the basis of knowledge in B). Whenever $\underline{B}X = \overline{B}X$, the collection of objects can be classified perfectly, and forms what is known as a crisp set. In the case $\underline{B}X$ is a proper subset of $\overline{B}X$, then the collection of objects contains some objects that cannot be classified with certainty, and the pair $(\underline{B}X, \overline{B}X)$ is called a rough set. Let's see an example. As the most common case is to synthesize definitions of the outcome (or decision classes) in terms of the conditional attributes A, we consider the set $X = \{x \mid Decision(x) = Accept\}$, as given by the table shown in Table 2.2. We then obtain the approximation regions $\underline{A}X = \{x5, x7, x8\}, \overline{A}X = \{x1, x2, x5, x7, x8\}$, and thus $BN_A(X) = \{x1, x2\}$ and $U - \overline{A}X = \{x3, x4, x6, x9\}$. It follows that the outcome *Decision* is rough since the boundary region is not empty. This is shown in Figure 2.4, where equivalence classes contained in the corresponding regions are shown.

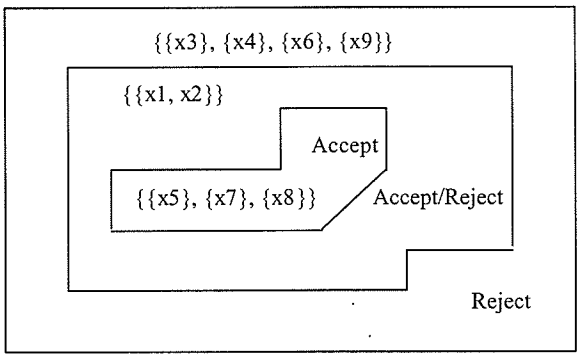


Figure 2.4 Approximating the set of hired people

2.3.2 Attribute Reduction and Decision Rules

Let $S=(U, A)$ be an information system. A *reduct* is a minimal subset $B \subseteq A$ such that $Ind(B)=Ind(A)$. In other words, a reduct is a minimal set of attributes from A that preserves the partitioning of the universe, and hence the ability to perform classifications. Thus, the reducts (of all the various types) serve the purpose of synthesizing minimal decision rules. Once the reducts have been computed, the rules are easily constructed by overlaying the reducts over the originating decision table and reading off the values. Any set of attributes has one or more reducts. The set of all reducts in S is denoted by RED(S).

We recall first two basic notions, the *discernibility matrix* and *discernibility function* which are essential in deriving decision system rules. Given an information system $S=(U, A)$ with n objects, the $n \times n$ matrix (c_{ij}) is called the *discernibility matrix* of S defined as follows:

$$c_{ij}=\{a \in A : a(x_i) \neq a(x_j)\}, \text{ for } i, j=1,...,n$$

A *discernibility function* $f_{M(S)}$ for information S is a Boolean function of m Boolean variables corresponding to attributes $a_1^*,...,a_m^*$ respectively, and defined as follows:

$$f_{M(S)}(a_1^*,...,a_m^*) = \wedge\{\vee c_{ij}^* \mid 1 \le j < i \le n, c_{ij} \ne \Phi\}, c_{ij}^* = \{a^* \mid a \in c_{ij}\}$$

Proposition 2.1 gives an important property which enables us to compute all the reducts of S.

Proposition 2.1 [43] Let $S = (U, A)$ be an information system, and let $f_{M(S)}$ be a discernibility function for S. Then the set of all prime implicants of the function $f_{M(S)}$ determines the set RED(S) of all reducts of S, i.e., $a_{i_1} \wedge ... \wedge a_{i_k}$ is a prime implicant of $f_{M(S)}$ if and only if $\{a_{i_1},...,a_{i_k}\} \in$ RED(S).

Below we present a procedure for computing reducts.

***Procedure* for computing RED(S)**[43]:

Input: An information system S.

Output: The set of all reducts in S.

Step 1. Compute the discernibility matrix for the system S.

Step 2. Compute the discernibility function $f_{M(S)}$ associated with the discernibility matrix *M(S)*.

Step 3. Compute the minimal disjunctive normal form of the discernibility function $f_{M(S)}$ (The normal form of the function yields all the reducts).

Let's see an example here. Table 2.3 is the hiring decision table shown in Table 2.2 with some modifications. The second row is removed to make sure that the table is consistent.

Moreover, the rows are reordered for convenience putting the accepted objects in the top rows.

	Diploma	Experience	French	Reference	Decision
y1(x1)	MBA	Medium	Yes	Excellent	Accept
y2(x5)	MSc	High	Yes	Neutral	Accept
y3(x7)	MSc	High	Yes	Excellent	Accept
y4(x8)	MBA	High	No	Good	Accept
y5(x3)	MBA	Low	Yes	Neutral	Reject
y6(x4)	PhD	Low	Yes	Good	Reject
y7(x6)	MSc	Medium	Yes	Neutral	Reject
y8(x9)	PhD	Low	No	Excellent	Reject

Table 2.3 *Hiring*: A reordered decision table.

Applying the above procedure for computing reducts to this table we obtain the symmetrical discernibility matrix *M(S)* presented in Table 2.4.

U	y1	y2	y3	y4	y5	y6	y7	y8
y1	Φ							
y2	Φ	Φ						
y3	Φ	Φ	Φ					
y4	Φ	Φ	Φ	Φ				
y5	E, R	D, E	D, E, R	E, F, R	Φ			
y6	D, E, R	D, E, R	D, E, R	D, E, F	Φ	Φ		
y7	D, R	E	E, R	D, E, F, R	Φ	Φ	Φ	
y8	D, E, F	D, E, F, R	D, E, F	D, E, R	Φ	Φ	Φ	Φ

Table 2.4 *Hiring*: The discernibility matrix

Thus the discernibility function $f_{M(S)}$ can be presented as follows (where the one-letter Boolean variables correspond to the attribute names in an obvious way):

$$f_{M(S)} = (E \vee R) \wedge (D \vee E \vee R) \wedge (D \vee R) \wedge (D \vee E \vee F)$$
$$\wedge (D \vee E) \wedge (D \vee E \vee R) \wedge E \wedge (D \vee E \vee F \vee R)$$
$$\wedge (D \vee E \vee R) \wedge (D \vee E \vee R) \wedge (E \vee R) \wedge (D \vee E \vee F)$$
$$\wedge (E \vee F \vee R) \wedge (D \vee E \vee F) \wedge (D \vee E \vee F \vee R) \wedge (D \vee E \vee R)$$

After simplification (using the absorption laws) we get the minimal disjunctive normal form as follows:

$$f_{M(S)} = E \wedge (D \vee R) = (E \wedge D) \vee (E \wedge R)$$

There are two reducts:

$$R_1 = \{D, E\} = \{Diploma, Experience\}$$

and

$$R_2 = \{E, R\} = \{Experience, reference\}.$$

Thus RED(S) $= \{R_1, R_2\}$.

Generally, a method used to find a proper subset of attributes of A with the same classificatory power as the entire set A is termed *attribute reduction* [44].

2.4 Summary

Computational Intelligence has become an important field in computing. It includes a family of problem-stating and problem-solving methods that attempt to mimic intelligence. Neural networks, evolutionary computing, fuzzy logic and rough sets constitute the backbone of its domain. In this chapter, the key features for neural networks and evolutionary computing are briefly discussed and fuzzy logic and rough sets are then discussed in more detail.

Chapter 3

Literature Review

Computational Intelligence techniques have been proposed and in some cases used for solving control problems in telecommunication networks (IP, ATM, Mobile networks and Active networks). In this chapter, we give a thorough review of current research efforts with focus on applications of fuzzy logic in telecommunication networks.

3.1 Fuzzy Logic in Telecommunication Networks

The discipline of fuzzy logic, fuzzy systems, and fuzzy modeling has witnessed its greatest success in real-world automatic control applications, including subway control, autonomous robot navigation, autofocus cameras, image analysis, and diagnosis systems. Application of fuzzy logic in telecommunication networks is less extensive than automatic control. A detailed search coupled with a thorough review of the literature reveals that current research in fuzzy logic in telecommunication networks extends from queuing, buffer management, connection admission control, and traffic control to flow control, congestion control, routing, policing, bandwidth allocation, network management, and quantitative performance evaluation of networks. To facilitate comprehension, the fuzzy literature may be organized into three efforts – modeling and control, management and forecasting, and performance estimation.

Bonde and Ghosh [45] present the use of fuzzy logic to model buffer queues in cell-switching networks. They introduce the concept of fuzzy thresholds toward robust, adaptive buffer management in sharp contrast to the traditional, abrupt, inflexible, binary thresholds. They demonstrate that fuzzy thresholds cause the buffer queue to exhibit "soft" behavior, i.e., greater ability to adapt to dynamic conditions, and robustness, i.e., resilience to rapid dynamic changes in network traffic, and favorably impact cell discarding. Razouqi *et al.* [46] develop a model of fuzzy threshold-based buffer management for a 50-switch representative cell-switching network to study its performance under realistic conditions. They also present an approach to reroute to their final destinations, the fraction of the selectively blocked cells that correspond to the difference of cell loss due to buffer overflow between the fixed and fuzzy schemes. Moreover, they report on the influence of the buffer management scheme on the end-to-end delay performance of a representative cell-switching network. Ascia *et al.* [47] propose a fuzzy buffer management mechanism for packet switched networks, namely ATM and IP networks. The proposed mechanism is able to guarantee the QoS requirements of high-priority traffic flow, allowing at the same time the exploitation of unused buffer resources to accommodate low-priority traffic flow in order to maximize the total throughput. They demonstrate that their fuzzy scheme out-performances other traditional logic based mechanisms, such as threshold and push out mechanisms (The push out algorithm works as follows: when the queue is not full, every incoming cell is accepted regardless of its priority; when the queue is full and a low-priority cell arrives, it will be discarded; when the queue is full and a high-priority cell arrives, it can push out a low-priority cell if there is one in the queue, or the arriving cell will be discarded

otherwise). Uehara *et al.* [48] and Ren *et al.* [49] both present a fuzzy connection admission control method for ATM networks. The method proposed by Uehara *et al.* [48] is based on the possibility distribution of the cell loss ratio. The possibility distribution is estimated in a fuzzy inference scheme by using observed data of cell loss ratio. This method makes possible secure connection admission control, thereby guaranteeing the allowed cell loss ratio. Ren *et al.* [49] develop a dynamic connection admission controller (CAC) that supports cell loss requirements. The CAC algorithm explicitly computes the equivalent bandwidth required to support each class of connections based on on-line observations of aggregate traffic statistics as well as the declared parameters. They use Gaussian and diffusion approximations to characterize the aggregate traffic stream, and use fuzzy control strategy to combine model and measurement results to derive simple closed-form formulas to estimate the equivalent bandwidth in real time. They validate the proposed algorithms for various variable bit-rate traffic profiles and show that the system utilization can be substantially improved by appropriately tuning the fuzzy logic controller to combine traffic characteristics deduced from the declared parameters and traffic measurements.

The use of fuzzy techniques in traffic control is reported in Cheng and Chang [36], Ascia *et al.* [50] and Lim and Qiu [51]. Cheng and Chang [36] present the design of a fuzzy traffic controller that simultaneously manages congestion control and call admission control for ATM networks. The fuzzy traffic controller is a fuzzy implementation of the two-threshold congestion control method and the equivalent capacity admission control method extensively studied in the literature. They extract knowledge of conventional

control methods from numerous analytical data using a clustering technique and then use this knowledge to set parameters of the membership functions and fuzzy control rules. The obtained parameters of the membership functions and fuzzy control rules are then further calibrated through simulations using a genetic algorithm optimization technique. They report an 11% improvement in system utilization while maintaining the QoS contract comparable with that of the conventional equivalent capacity method. Ascia *et al.* [50] deal with a fuzzy logic-based traffic-source policing system which has been purposely designed to achieve real-time traffic control in ATM networks. They address two key issues related to the implementation of the fuzzy policer. The first focuses on the possibility of hardware implementation of the mechanism using VLSI technology; they present the design of a VLSI fuzzy processor which exhibits a level of performance of over three mega fuzzy logic inferences per second. The second issue concerns the suitability of applying the fuzzy policer to the policing of several classes of sources to reach high levels of cost effectiveness and scalability. Lim and Qiu [51] propose a fuzzy logic traffic control scheme which uses a fuzzy logic target utilization factor generator with fuzzy logic prediction techniques. A key feature of the proposed controller is its ability to target link utilization dynamically based on the predicted future buffer condition in the switch. The proposed controller has stable and robust operations under the circumstance of long round-trip delays. The control scheme is applied to both approximate and exact fair rate computation switch algorithms. Pitsillides *et al.* [52] describe a fuzzy explicit rate marking (FERM) traffic flow control algorithm for a class of best effort service, known as available bit rate (ABR), proposed by the ATM Forum. FERM is an explicit rate-marking scheme in which an explicit rate is calculated at the

ATM switch and sent back to the ABR traffic sources encapsulated within resource management cells. The flow rate is calculated by the fuzzy congestion control module by monitoring the average ABR queue length and its rate of change, then by using a set of linguistic rules. On the condition that the number of sources and the round trip delay time are both small, FERM exhibits a robust behavior, even under extreme network loading conditions, and ensures fair share of the bandwidth for all virtual channels regardless of the number of hops they traverse. Additionally, compared with the enhanced proportional rate control algorithm (a current favorite by the ATM Forum), FERM controls congestion substantially better, offers faster transient response, leads to lower end-to-end delay and better network utilization. Kuan and Andrew [53] further investigate the performance of FERM in the presence of large round trip times. In this case, the fuzzy logic scheme is found to be significantly more sensitive to increase in the round trip time than a simple threshold based scheme. Hu and Petr [54] present the design and analysis of an end-to-end rate-based feedback flow control algorithm motivated by the ABR services in ATM networks. Their approach is to first predict the ABR buffer status, then base fuzzy-logic rate control decisions on these predicted values, and finally tune the controller parameters using gradient descent methods. They demonstrate that this predictive self-tuning fuzzy-logic control scheme is efficient, stable, and outperforms other proposed ABR rate controller in a variety of network environments. To relieve congestion and achieve high server utilization in ATM-based networks, Pitsillides *et al.* [55] utilize fuzzy logic control techniques to develop a novel backward congestion notification scheme fuzzy backward congestion notification (FBCN). They state that the complexity of the ATM networks and their dynamic parameters renders their analytic modeling very difficult if not impossible.

They show that the FBCN scheme keeps the queues well controlled, not only minimizes the cell loss for even offered network loads exceeding 100% for the case of both an ATM WAN and ATM LAN model but also keeps utilization close to unity. A congestion avoidance scheme that makes use of fuzzy set theory is described by Qiu [56]. He proposes a fuzzy logic predictor to estimate the output queue length. This information together with current queue length and growth rate is provided to a fuzzy inference system to generate a rate factor. This factor can be used alone or in conjunction with other algorithms to calculate ABR traffic bandwidth allocation and is ultimately influential in modifying the ER field in BRM cells. He shows that the incorporation of a predictor improves QoS performance in high-speed networks when propagation delay is significant, and that the proposed fuzzy logic predictor works better than a traditional auto-regression predictor. Chen *et al.* [57] propose a fuzzy autoregressive model to describe the traffic characteristics of high-speed networks. The proposed model has excellent capability to provide precise prediction for efficient congestion control and thus has significant potential for practical network traffic control design. The validity of this model is verified using actual traffic data in Bellcore LAN network and from multimedia broadband networks.

Two applications of fuzzy logic techniques to traffic routing are described by Chemouil *et al.* [58] and Vasilakos *et al.* [59]. Chemouil *et al.* [58] present a fuzzy control approach for adaptive traffic routing. They show that the proposed fuzzy control approach could provide an effective framework for robust control of traffic routing in telephone networks. Vasilakos *et al.* [59] provide an early investigation of the applicability of

computational intelligence (CI) techniques in the active networking context. They report on the characteristics of the computational intelligence technologies and describe a novel approach to routing in active networks. The proposed solution to the routing problem involves the following components: *Roaming Agents*, which are moving from element to element to collect and distribute information on network state; *Routing Agents* which, at each network element, are responsible for spawning roaming agents and are the recipients of the information collected by them and also act as resource brokers for connectivity; a *CI engine*, which is a set of active extensions that include several subcomponents that form a generic library-like algorithmic infrastructure. The components implemented for the CI engine include an *evolutionary-fuzzy controller* which clusters paths according to their current state characteristics, a *stochastic reinforcement learning automation* which, given a set of paths, computes the "best" route for a given set of connection constraints, and an *evolutionary-fuzzy time series predictor* that can be used for predicting traffic load on network links based on past link utilization information. While its potential performance gains can not be quantified at this initial research stage, the proposed approach is proved to be able to provide an alternative solution with interesting properties that were unthinkable in the traditional model. Catania *et al.* [60] propose a fuzzy policing mechanism for ATM networks to detect violations of the negotiated parameters while reducing the probability of false alarm. The mechanism monitors the number of cells transmitted by the user since connection and utilize fuzzy rules to increase the threshold for conformance and lower it during periods of nonconformance. A high-speed network will typically assign each admitted connection a fixed bandwidth, somewhere between its mean and peak transmission rate. As an alternative, Slonowsky *et al.* [61]

develop a novel scheme for re-assigning network resources periodically on the basis of current network conditions. Their algorithm, which employs the theory of fuzzy logic control, is computational efficient, making frequent bandwidth re-allocations over relatively small time intervals feasible. In situations where traffic intensities change drastically over short time intervals, their algorithm, tested against both static and non-fuzzy bandwidth re-allocation schemes, significantly lowers data loss while increasing network efficiency. Jiang *et al.* [62] investigate algorithms, based on the theory of fuzzy logic systems, which use on-line traffic measurements to adaptively learn packet arrival patterns, leading to "model-free" traffic prediction. The main predictor, and several novel variants thereof are developed and tested on diverse data streams. For autoregressive-type traffic sources, their algorithms are on par with standard time-series predictors. More significantly, their algorithms provide reliable predictions for highly variable, non-stationary MPEG data; a situation where standard methods are poorly suited. Jensen [63] presents a fuzzy logic control mechanism for ATM network management. He observes that fuzzy logic serves as a better tool to facilitate knowledge representation and inferencing. The presented fuzzy logic controller is shown to be a suitable instrument to manage the occurrence of permanently changing traffic situation considering the negotiated parameters of each connection. It has the ability to provide good service quality characterized by low losses, even if the traffic load changes permanently.

Edwards and Sankar [64] report encouraging results in a fuzzy algorithm driven hand-off operation for a cellular system where the cell size is reduced to 10-100 m. The algorithm combines the strength of the received signal with distance measurements to yield a hand-

off factor, which decreases monotonically as a mobile unit moves away from the base station, thus tracking the actual signal closely. Edwards and Sankar [65] also present a predictive fuzzy hand-off algorithm which is capable of providing high performance for line-of-sight (LOS) and non-line-of-sight (NLOS) micro-cellular conditions. When compared to current hand-off algorithms based on signal averaging and hysteresis margin that are utilized in TDMA/FDMA systems, the predictive fuzzy algorithm provides a superior performance. Lau *et al.* [66] report success at reducing the number of hand-offs through a fuzzy algorithm that dynamically adjusts the signal averaging interval and the hysteresis threshold. Shum and Sung [67] introduce a fuzzy layer selection method for a two-layer cellular system with fixed channel assignment. In this method, the channel occupancies in the target microcell and macrocell, as well as the cell dwell time, are used as inputs to the fuzzy system. The fuzzy rules are constructed by common sense. Compared to the traditional threshold method, the fuzzy approach can reduce the number of hand-offs by at least 10% given other conditions constant. Levin [68] presents a computation and performance analysis of computer networks with nondeterminate fuzzy time parameters.

3.2 Summary

This chapter overviewed a sample of research incorporating fuzzy logic techniques in the telecommunication area.

A detailed search coupled with a thorough review of the literature revealed that current research in fuzzy logic in telecommunication networks can extend from queuing, buffer management, connection admission control, and traffic control to flow control,

congestion control, routing, policing, bandwidth allocation, network management, and quantitative performance evaluation of networks. The fuzzy literature was organized into three efforts – modeling and control, management and forecasting, and performance estimation, in order to facilitate comprehension.

Chapter 4

Improved Data Transport Using UFTP

4.1 Motivation for Designing a New File Transfer Protocol

The File Transfer Protocol (FTP) [69] has been the most popular file transfer protocol in the Internet. However, experience from Internet users and results from recent research have shown that the data transfer speed of FTP can be very slow, especially when transferring 'big' files across a WAN [70]. The slow performance of FTP, which is largely due to the inefficiency of its underlying Transport Control Protocol (TCP) [71], has motivated us to envisage a new protocol for transferring files over the Internet.

TCP is the most commonly used transport protocol in the Internet. It uses an implicit feedback based window flow control mechanism. The window size is increased on receipt of every acknowledgement and decreased on detection of a packet loss. This algorithm is based on the assumption that losses are always due to buffer overflow at some intermediate store-and-forward gateway, thus indicating congestion in the network. But experiments conducted over the Internet have shown that this assumption is not true [22]. It is observed from these experiments that losses are not always caused by buffer overflows resulting from network congestion. There are diverse hardware and software problems with many gateways in the Internet that may result in packet loss even when

there is no congestion. The results presented in [23] also show that even at a quite low likelihood of non-congestion related loss (1%), the throughput of a connection reduces significantly when this flow control algorithm is used. Apart from this, TCP repeatedly induces packet loss as it probes to identify available bandwidth via an "additive increase, multiplicative decrease" congestion control mechanism [24]. Although this approach works well over current local-area networks (LANs) and even some WANs, it fails miserably as the bandwidth-delay product of a network connection becomes large.

There are two ways to solve the problem of the poor performance of TCP: modify TCP or avoid TCP. In fact, numerous modifications to TCP have been proposed in the literature in order to improve the performance of TCP. Some of them are as follows:

- Round-trip time measurement [72]
- Window scale option [73]
- Selective acknowledgement [74]
- Start-up behavior improvement [75]

Some application-level protocols choose to avoid TCP in order to achieve better performance. The Trivial File Transfer Protocol (TFTP) [76] takes advantage of the User Datagram Protocol (UDP) [77] as its transport-level service to manipulate files with no reliability guarantee. TFTP is a simple file transfer protocol implemented on top of UDP. It is designed to be small and easy to implement. Therefore, it lacks most of the features of regular FTP. With TFTP, the file to be transferred is sent in fixed length blocks of 512

bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent. If a packet gets lost in the network, the intended recipient will timeout and may retransmit his lost packet, thus causing the sender of the lost packet to retransmit that lost packet. The sender has to keep just one packet on hand for retransmission, since the lock step acknowledgment guarantees that all older packets have been received. Both machines involved in a transfer are considered senders and receivers. One sends data and receives acknowledgments, and the other sends acknowledgments and receives data.

The File Segment Transfer Protocol (FSTP) [70] is another file transfer mechanism. In this protocol, a TCP socket is used for sending commands (control channel), and a UDP socket for transmitting data (data channel); whereas in FTP, two TCP sockets are used for the same purpose. Unlike TCP, UDP does not provide reliable connections. Thus the integrity of data is provided in the application level, i.e., FSTP itself. FSTP pre-optimizes the flow rate of data by introducing an inter-packet transmission delay whose value is estimated before any data is sent and then held constant in the later data transferring. Unlike TFTP, FSTP doesn't use lock step acknowledgments; instead, the FSTP server sends the whole set of data to the FSTP client without waiting for any acknowledgment. Then the FSTP server retransmits the lost packets until all the data is received. FSTP takes advantage of the more reliable network infrastructure used in today's Internet than when TCP was first introduced many years ago.

In the following we will propose a new UDP-based File Transfer Protocol (UFTP). UFTP is an application level protocol which is also implemented on top of UDP. It is the enhancement and modification of the FSTP protocol, in which a dynamic flow control algorithm using rough or fuzzy sets as well as a "credit-granted" acknowledgement scheme will be introduced. It is designed to transfer data much faster than FTP and thus reduce latency for Internet users.

4.2 The UDP-based File Transfer Protocol (UFTP)

To better understand the UFTP protocol, consider what kinds of network connections that are used for an FTP session and compare them to those used in our new protocol. The FTP protocol uses two TCP connections, one for exchanging command/control packets, and the other for the data itself. Basically, the FTP protocol is not concerned with retrieving missing/corrupted packets. The job of "providing a reliable network connection" is delegated to the transfer layer protocol TCP. TCP ensures the integrity of the data by checking the incoming packets, and asking the sender for retransmission of the erroneous/missing packets. UFTP operates differently from FTP. Although UFTP still has a TCP network connection to send command/requests to the server, all of the data is transferred via UDP packets.

Sending data over UDP doesn't bind us to the restrained performance of TCP, and there is less processing overhead compared to a TCP connection. However, since UDP only

provides a datagram service, the necessary functionality for data consistency is provided at another level, i.e., the application level, by UFTP itself. To realize this functionality for data consistency, UFTP attaches a unique sequence number to each data packet to be sent.

UFTP uses "credit-granted" acknowledgements to maintain the integrity of the data. Unlike FSTP, in order to avoid too many packets being placed on the Internet at a time, which may cause network congestion and slow down the network throughput, the UFTP server is granted a credit of packets that can be sent without having to wait for acknowledgements. This would allow the UFTP server to fill the pipe, then, the arrival of acknowledgements would maintain and adjust a continuous flow of data. TFTP is thus a particular case of such protocols, where a credit of one packet is given to the data-packet sender. Specifically, the UFTP "credit-granted" scheme works like this. The whole set of data packets to be transferred is divided (softly) into many *sections* based on the credit granted to the UFTP server. One section is sent at a time. The UFTP server sends out the first section. The UFTP client receives packets until a timeout occurs. Then the client checks its own packet-tracking table to identify the missing packets. A request of retransmission of the missing packets is then sent back to the UFTP server. The server retrieves the missing parts of the section from its local disk, and sends them again to the client. This loop continues until the whole section is completely received by the client. Then UFTP transfers the next section, and then the next section, until all the sections, i.e., the whole file is successfully transferred to the user.

Obviously, lots of the packets would get lost in transit if we transmit these packets at the maximum possible speed, due to a relatively smaller maintainable bandwidth over the Internet as well as buffer overflow on the client machine. Thus, to maintain a reasonably small percentage of packet loss, an inter-packet transmission delay is added in our protocol design. Adding this delay results in a greater packet reception success rate (PRSR) and much fewer retransmission requests. This inter-packet transmission delay also works to minimize network traffic and avoid network congestion.

Using UDP as the transport level protocol implies the following:

- The UDP header doesn't contain any field to hold the sequence number of the data packet (as in TCP). Therefore, the upper layer protocol UFTP should have a header that contains the sequence number. This number is used to keep track of which packet has been received and pinpoint the missing ones. In our design, a UFTP header is attached to the data to be transmitted. The UFTP header (Fig. 1) consists of an identifier and the sequence number. The identifier is chosen as the name of the file from which the data is taken. An asterisk (*) is appended to this field as a flag indicating the end of the field. The size of the sequence number field is not fixed. It is calculated dynamically for each UFTP session, based on the size of the file to be transmitted and the agreed-upon UDP packet size.

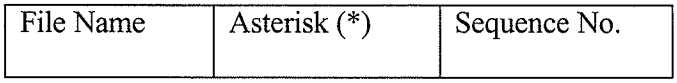


Figure 4.1 UFTP header

- The UDP header is much smaller than TCP. Thus, less processing and network overhead per packet is required.
- To keep the fragmentation overhead as low as possible, we have to choose the UDP packet size to be less than MTU (maximum transfer unit) size for the network through which the data is passing. A default value of 1500 bytes is chosen in our design (it seems to be the optimal value for the packet size given the fact we are not behind firewalls and are on Ethernet LANs. This is also a parameter that can be easily adapted and maximized although not the topic of this research).

The operations involved in a UFTP file transmission are as follows:

1. The UFTP client opens up a TCP connection with the server for exchanging commands.
2. The packet size is set by the client and forwarded to the server.
3. A brief test is conducted between the server and the client in order to calculate an initial value for the inter-packet delay. This parameter is directly related to a bandwidth-delay product which could also be estimated automatically if required.
4. The client requests a file listing over the TCP connection. The server returns a list of available files and their respective sizes.
5. Based on the size of the file to be transferred and the credit granted to the server, the client determines the number of "sections" that the file is divided into. Each section is further divided into many "units". For example, a "section" might be 10Mbyte, and a "unit" 2Mbyte.

6. The client issues a "SEND" command to the server to transfer the first "unit" of the file to be retrieved, taking note of the file's size from the previously acquired file listing.
7. The server opens a UDP connection with the client and starts sending the first "unit" of the file as a stream of UFTP packets, separated by the initial inter-packet transmission delay (step 3) such that one packet is transmitted for each previously specified delay interval. The UFTP header of each packet contains the file name and a sequence number to indicate the position of the data in the file.
8. The client stores the incoming UFTP packets and keeps track of the ones it received until a timeout occurs.
9. The client operates its rough controller to get an optimal value for the inter-packet transmission delay, and then sends it to the server.
10. Using the new inter-packet transmission delay, the server sends the next "unit" without doing retransmission of the missing/corrupted packets in the previous unit(s).
11. Step 8, 9 and 10 is repeated until the last "unit" of this "section" is sent out.
12. The client generates a list of missing/corrupted packets within this "section" and sends them, over the TCP connection, in a "retransmission" request to the server.
13. The server retransmits the requested UFTP packets in the same format as the original transmission, using the most recently obtained inter-packet transmission delay from the client.

14. This retransmission continues until all of the packets of this "section" are completely received by the client. Then the client issues a "SEND" command to the server to send the first "unit" of the next "section".
15. Using the optimal value of the inter-packet transmission delay obtained from the transmission of the previous "section", the server sends the requested "unit".
16. Step 8-15 is repeated until the last "section" of the file is completely received.
17. The client extracts the original data from the received packets and writes the transferred file to disk (this may involve sorting packets to restore the proper packet order). This aspect is not crucial to demonstrating the rough controller as a more efficient random access file building process can also be employed. The whole file transfer is then complete.

Figure 4.2 is the flow chart of UFTP operations, where M is the number of "sections" of the file to be transferred and N_i is the number of "units" of Section i.

4.3 UFTP Dynamic Flow Control Using Rough Sets

As stated above, UFTP introduces an inter-packet transmission delay to accomplish the flow control. Thus setting up a proper value for the inter-packet transmission delay is very important. Too big a value means too slow a speed for each transfer, even though the packet reception success rate may be high. Too small a value will cause lots of packets to be lost in each transfer, which will result in too many retransmissions. In UFTP, the initial value of this delay is obtained through the measurement of the throughput between the client and the server by conducting a brief test which takes place

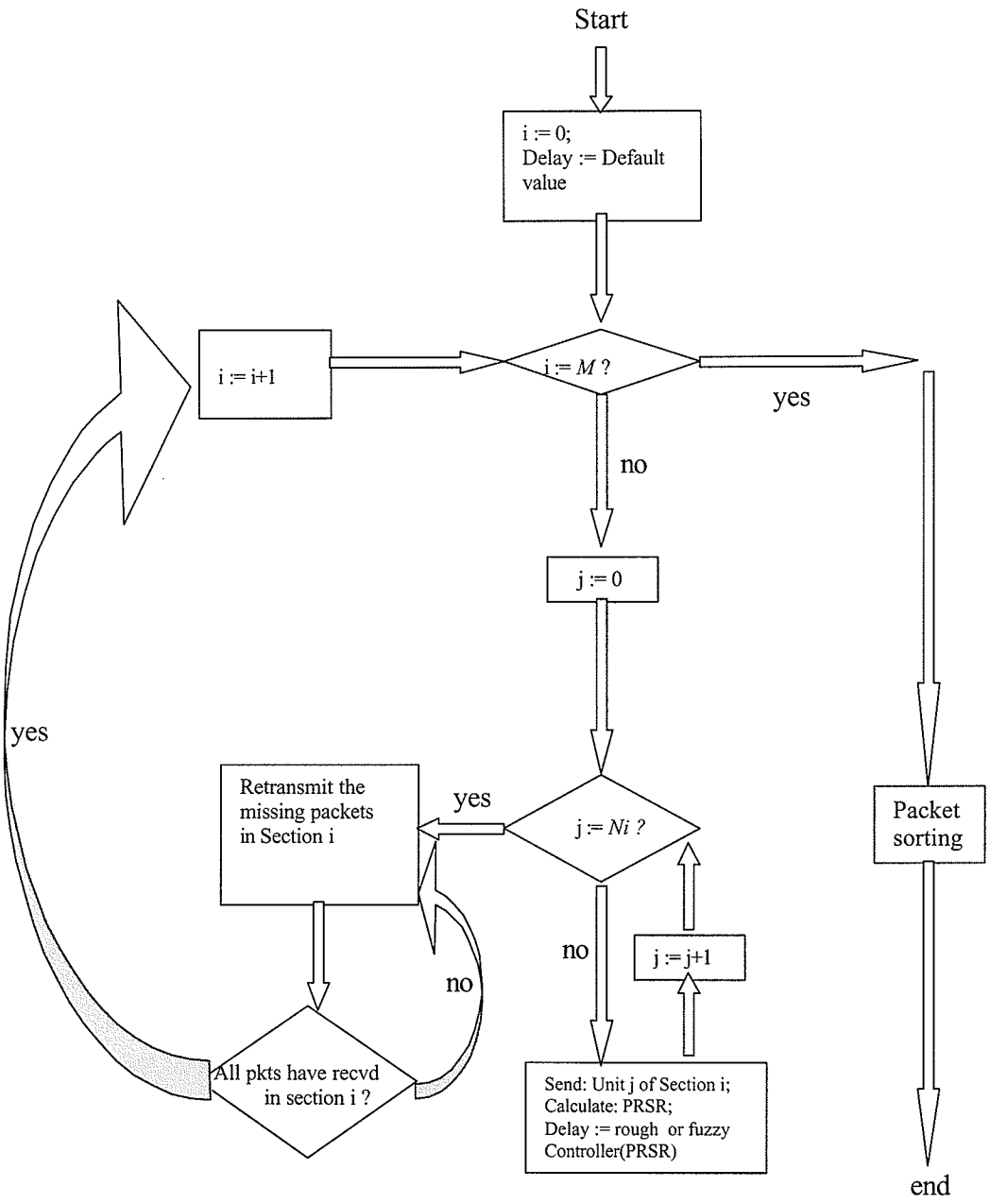


Figure 4.2 Flow chart of UFTP operations

over a very short time interval. The test method used here follows that in FSTP. It's very simple although not efficient and works in this way. The server attempts to flood its

connection with some UDP packets destined for the client. The client then calculates the packets it received in total together, with the noted transmission time from the server, and calculates an appropriate delay. For example, if the server sends 1,000 packets in one second and the client receives 100 packets, we can speculate that if we transmit a packet every 10ms, we should be able to maintain a relatively high packet reception rate of success while keeping the data transfer fast. It should be noted that obtaining a reasonable initial rate or delay is not the emphasis of this thesis.

Since the Internet traffic conditions change constantly from one moment to another, it is not a good idea to just use a fixed value of the inter-packet transmission delay to transfer a very large file, no matter how "good" the value is estimated at the beginning of the file transfer process. Thus better performance can be expected if a dynamic flow control mechanism is implemented.

In this section, we propose a rough set based dynamic flow control algorithm for UFTP. Rough control methods have been successfully applied in many applications [78-84]. They provide a means of deriving rules from decision system (data) tables. The derived rules are then used as a basis for evaluating as well as controlling processes. As in the design of the rough controller for UFTP, derived rules are used to select correction factors to adjust the inter-packet transmission delay towards its optimal value from time to time.

4.3.1 Design of the rough controller

To begin this approach, it is first necessary to design *an information table* (selection of universe of *objects* and selection of *attributes*) based on the characteristics of the flow control. This design is accomplished by selecting a universe consisting of objects that are observations of *packet reception success rate* (*PRSR*) (for the reason of simplicity, we consider only one feature in our controller design; more features may be added in future research). Each *PRSR* is the number of data packets successfully received by the client over the number of data packets sent by the server during each observation period. That is, our sets of objects (observations) are the *PRSR* of transmissions for different inter-packet transmission delays. In this research, *PRSR* is conceptualized in terms of fuzzy sets relative to linguistic labels *low*, *acceptable*, and *high*. For each object (observed *PRSR* of a transmission), we decide on correction factor (*dfactor*) for the inter-packet transmission delay (*IPTD*) to improve the network performance (i.e., latency reduction). The following formula is then used to calculate a new value for the inter-packet transmission delay:

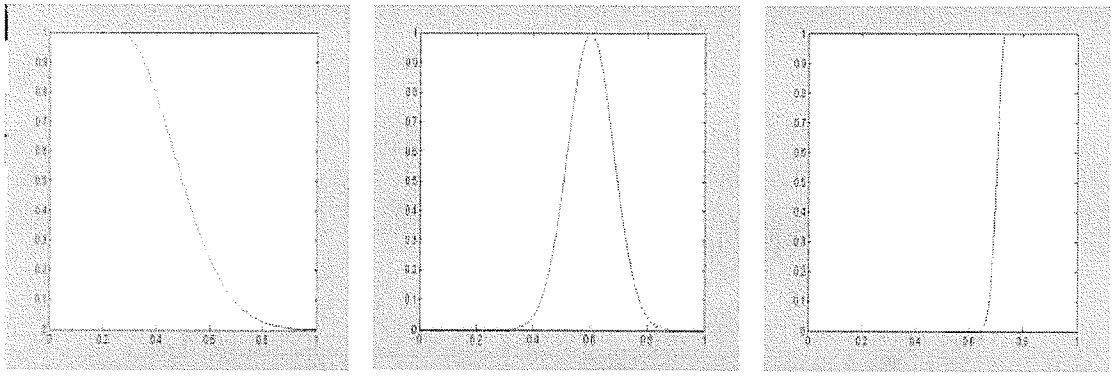
$$IPTD = IPTD * dfactor$$

Each decision value of *dfactor* is a judgment of the network performance from a calculated *PRSR*: the observed *PRSR* is compared to the performance of a "best-choice" *PRSR*. Based on the above consideration, a *decision table* is thus constructed with three conditional attributes: *PRSR_low*, *PRSR_acceptable* and *PRSR_high* for *granulations* of *PRSR* measurement, and one decision column *dfactor* as a correction factor for the *IPTD*. Sample rows from the rough controller tuning information table are given in Table 4.1.

PRSR∈ U\A		*PRSR_low*	*PRSR_acceptable*	*PRSR_high*	*dfactor*
x1	0.189	1	0.0	0.0	5.00
x2	0.382	0.830	0.025	0.0	1.92
x3	0.455	0.622	0.194	0.0	1.40
x4	0.560	0.325	0.883	0.0	1.00
x5	0.666	0.127	0.712	0.054	0.64
x6	0.710	0.080	0.389	0.752	0.50

Table 4.1 Sample decision table

The distribution of degree of membership values in a granule associated with the attribute *PRSR_low, PRSR_acceptable* or *PRSR_high*, is assumed to be approximately normal or semi-normal, with mean m (modal point) and standard deviation s (spread). Based on the data (observations) collected from our prototypical experiments, three membership functions in modeling the attributes *PRSR_low, PRSR_acceptable* and *PRSR_high* were obtained and are shown in Figure 4.3.



(a) *PRSR_low* (b) *PRSR_acceptable* (c) *PRSR_high*

Figure 4.3 Membership functions for the attributes

Once the decision table is completely constructed, we are ready to derive decision rules. Instead of using rough sets step by step to derive the rules, we will apply in our design a software tool *Rosetta* [85] which is implemented based on rough set theory. After deriving the reduct(s) {*PRSR_low, PRSR_acceptable*} with *Rosetta*, we are now faced with the modeling process. This process has the following three basic steps:

1. *Discretization*: Transforming non-categorical attributes in a decision table into categorical ones. The approach of rough sets requires only indiscernibility. This means that there is no need to define an order or distance when attributes of different types are combined. On the other hand, non-categorical attributes must be discretized in a pre-processing step. The discretization step determines how coarsely we want to view the world. Discretization is a step that is not specific to rough set approaches as most rule or tree induction algorithms also require them to perform well. The search for appropriate cut-off points essentially uses the approach of finding minimal Boolean expressions. Discretization is not an easy task, and its complete presentation is given in [86].
2. *Rule induction*: Synthesizing decision rules from a decision table. Dynamic reducts based on resampling approach is a successful method which is implemented in the *Rosetta* system. A sampling of the controller rules that are derived with *Rosetta* is given as follows:

   [PRSR_low([0.612, 0.654)) and PRSR_acceptable([0.004, 0.428))]

   or *[PRSR_low([0.694, 0.718)) and PRSR_aceptable([0.004, 0.428))]*

   $\Rightarrow$ *dfactor(1.4)*

Such rules are derived from a real-time decision system table based on a sufficient number of prototypical experimental measurements of the performance of file transfers and the granulation of these measurements.

3. *Rule application*: Applying the extracted decision rules to classify new cases. When a rough set classifier is presented with a new case, the rule set is scanned to find applicable rules, i.e., rules whose predecessors match the case. "If-then" statements in Java are used to implement the decision rules.

The basic structure of a rough controller can be illustrated as in Figure 4.4.

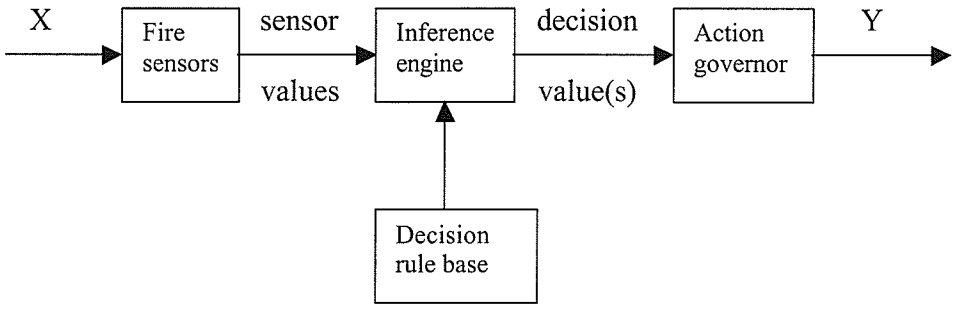


Figure 4.4 Basic Structure of a Rough Controller

4.3.2 Experimental Measurements

In the following, we use UFTPrough and UFTP to represent the protocols with the rough flow controller enabled and disabled, respectively. To evaluate the performance of UFTP with flow control using the designed rough controller and compare it with FTP, the following tests were conducted:

- Tests of different combinations of "unit/section"
- Performance tests under moderate traffic scenario
- Performance tests under heavy traffic scenario

The moderate traffic conditions are weekdays during school hours (9:00am – 5:00pm) and the heavy traffic conditions are generated by running FTP and UFTP at the same time. These two conditions are where FTP performances are usually very poor and thus the main concerns for this research.

A remote machine (Linux 2.4.4-2smp, Server) in Indonesia and a local machine (SunOs 5.8, Client) in University of Manitoba were used to conduct the tests of different combinations of "unit/section". The two machines are connected via a number of various speed connections. A large-size file with 20MB (23,725,050 bytes) was used in these tests.

One machine (SunOs 5.8, Server) in University of Victoria and another machine (SunOs 5.7, Client) in Lakehead University were used to conduct the UFTPrough performance tests under both moderate and heavy traffic scenario. The two machines are connected via high-speed Internet connections. Three files with 15MB, 30MB and 100MB were used in the tests for performance comparison illustrated here.

A. Tests of different combinations of "unit/section"

The test results are shown in Figure 4.5. They are the averages of 8 transfer times for each "unit/section" combination. From the test results, it seems that the combination "unit = 1400 packets, section = 7000 packets" is the best for UFTP performance. This means the whole set of data packets is effectively partitioned into many "sections" of 7000 packets (about 10Mbyte) each with 5 "units" of 1400 packets (about 2Mbyte). These tests although not conclusive illustrate that the number of "sections" and size of "units" play a role in the granularity of the adaptation and optimization of them is possible.

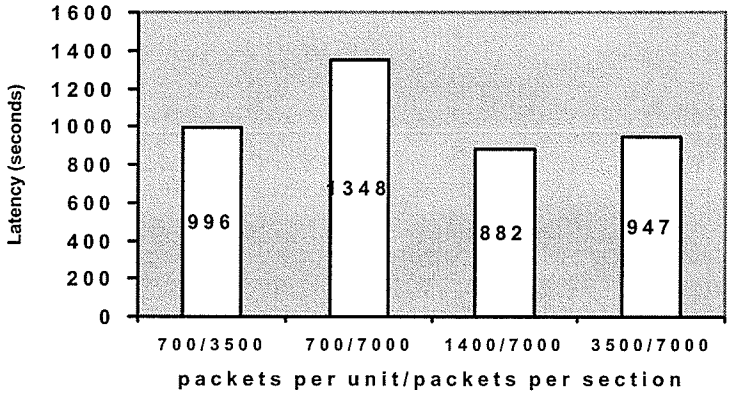


Figure 4.5 Results of UFTP tests for different combinations of "unit/section"

B. Performance tests under moderate traffic scenario

Figure 4.6, Figure 4.7, and Figure 4.8 are the measurements of transfer times for downloading the three different-size files under moderate traffic conditions. The measurements for each file are repeated 30 times. From the test results, we see that

UFTPrough transfer time is approximately 1.20 times faster than FTP. Also from the test results, we see that UFTPrough transfer time is approximately 1.15 times faster than UFTP when downloading the 15MB file, 1.17 times faster than UFTP when downloading the 30MB file, and 1.12 times faster than UFTP when downloading the 100MB file, which indicates that the rough flow controller works as anticipated.

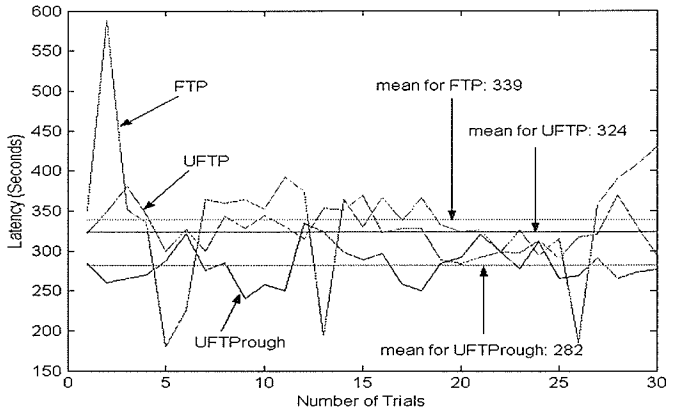


Figure 4.6 Downloading a 15MB file with rough controller under moderate traffic scenario

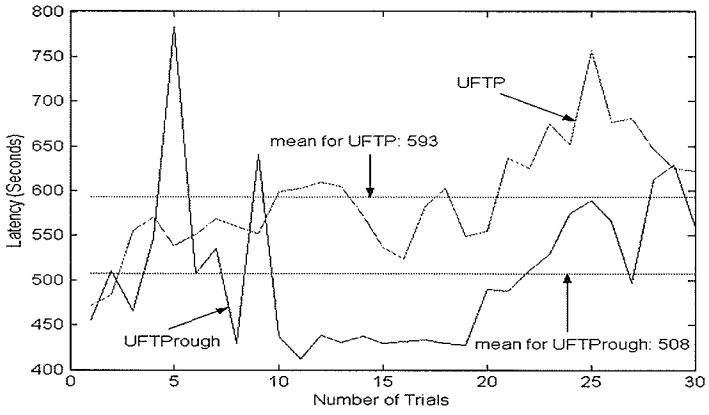


Figure 4.7 Downloading a 30MB file with rough controller under moderate traffic scenario

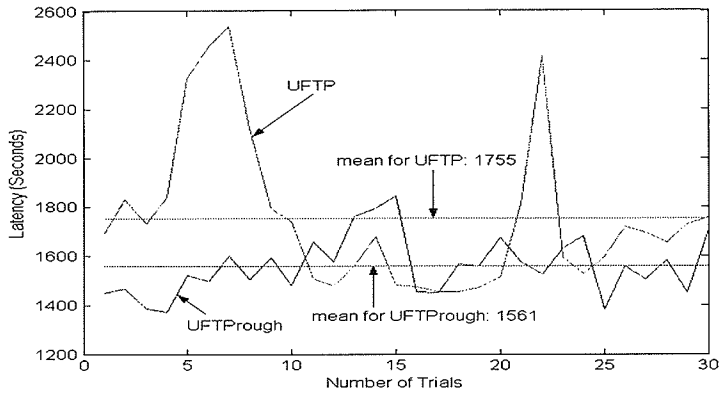


Figure 4.8 Downloading a 100MB file with rough controller under moderate traffic scenario

C. Performance tests under heavy traffic scenario

Figure 4.9, Figure 4.10, and Figure 4.11 are the measurements of transfer times for downloading the three different-size files under heavy traffic conditions. The measurements for each file are also repeated 30 times.

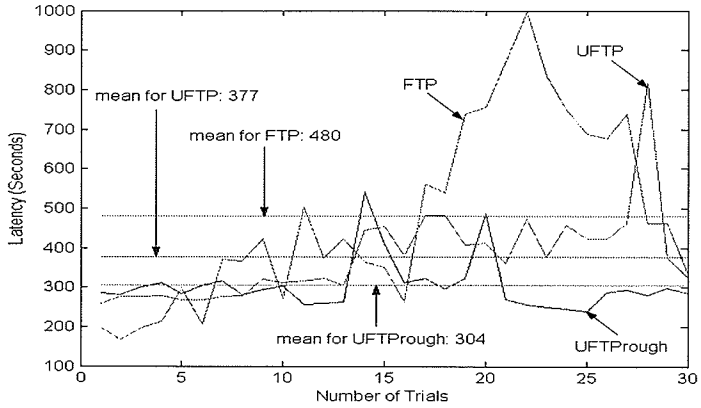


Figure 4.9 Downloading a 15MB file with rough controller under heavy traffic scenario

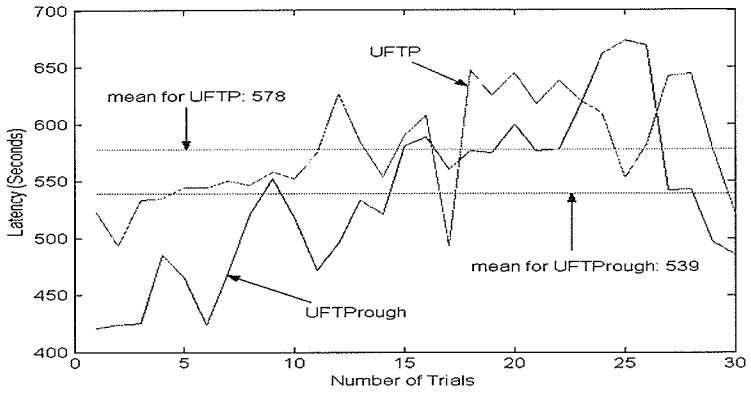


Figure **4.10** Downloading a 30MB file with rough controller under heavy traffic scenario

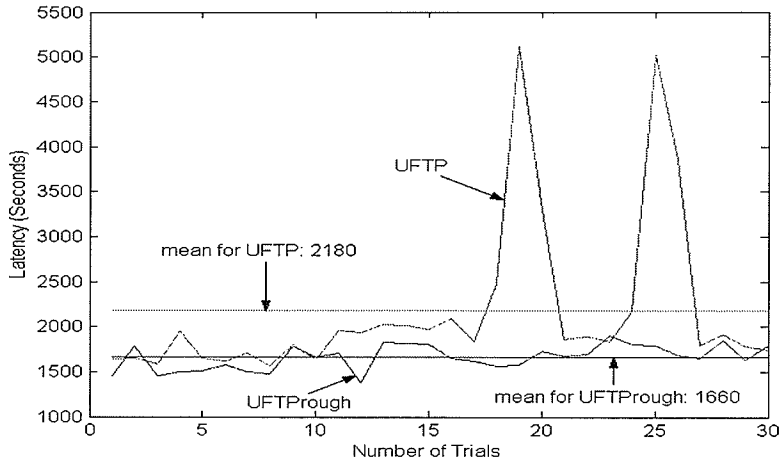


Figure 4.11 Downloading a 100MB file with rough controller under heavy traffic scenario

From the test results, we see that UFTPrough transfer time is approximately 1.58 times faster than FTP. Also from the test results, we see that UFTPrough transfer time is approximately 1.24 times faster than UFTP when downloading the 15MB file, 1.07 times

faster than UFTP when downloading the 30MB file, and 1.31 times faster than UFTP when downloading the 100MB file, which indicates that the rough flow controller works as anticipated.

These test results also indicate that when transferring a file using UFTPrough (under both moderate and heavy traffic conditions), the larger the file to be transferred is, the more significant the file transfer throughput improvements are. They also indicate that UFTPrough throughput improvements as compared to FTP are more dramatic under heavy network traffic conditions.

4.4 UFTP Dynamic Flow Control Using Fuzzy Logic

In this section, we propose a fuzzy logic based algorithm as an alternative method for UFTP flow control.

4.4.1 Design of the fuzzy logic controller

Designing a fuzzy logic controller involves selection of suitable mathematical representations for t-norm, s-norm, defuzzification operators, fuzzy implication functions, and shapes of membership functions among a rich set of candidates. Particular selection of these operators and functions alter the nonlinear input-output relationship, or in other words, the behavior of a fuzzy logic controller. But research has shown that same effects

can be achieved by proper modification of the rule base. Therefore, in practical applications, usually computationally lighter and well-studied operators and functions are selected, and desired behavior of a fuzzy logic controller is obtained by altering the rules.

The fuzzy flow controller is a fuzzy logic controller that deals with the flow control related procedure. Two input linguistic variables are chosen here: the *packet success rate* (*psr*) which is the number of data packets successfully received by the client over the number of data packets sent by the server during each observation period, and the *change of packet success rate* (*cpsr*) which is the variation of the current *psr* compared to last *psr*. The output linguistic variable is the *fractional delay rate* (*fdr*) which is the correction factors of the inter-packet transmission delay.

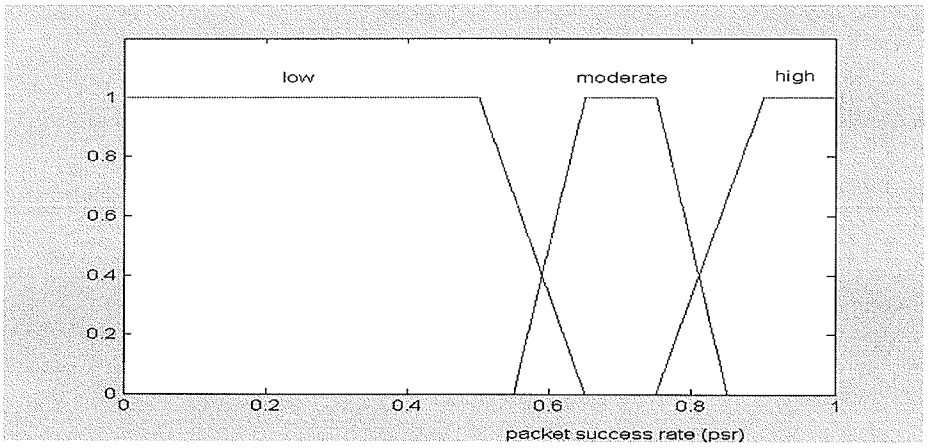
In the fuzzifier, we determine term sets at the right level of granularity for describing the values of linguistic parameters. The number of terms in a term set is selected as a compromise between the complexity and the controlled performance. We assign more terms to the term sets *cpsr* and *fdr* than to the term set of *psr* because we expect that both *cpsr* and *fdr* contain more information than *psr*. Accordingly, the term sets of *psr*, *cpsr* and *fdr* are defined respectively as follows:

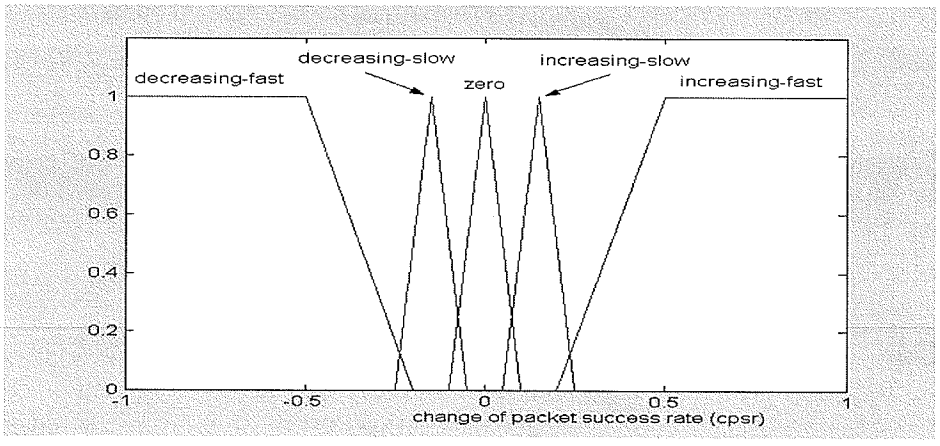
T(*psr*) = {low, moderate, high};

T(*cpsr*) = {decreasing_fast, decreasing_slow, zero, increasing_slow, increasing_fast};

T(*fdr*) = {very_little, little, moderate, high, very_high};

Next, we define a membership function for each term set. The main task is to choose a function with proper shape and position. As in most applications, here we use the shape of triangular and trapezoidal functions. Figure 4.12 shows the membership functions we have chosen for *psr*, *cpsr* and *fdr* respectively. Note that, in our application, the center (or the edge) and the width of the triangular or trapezoidal functions for each term set are specified based on our knowledge about the system and the performance requirement.





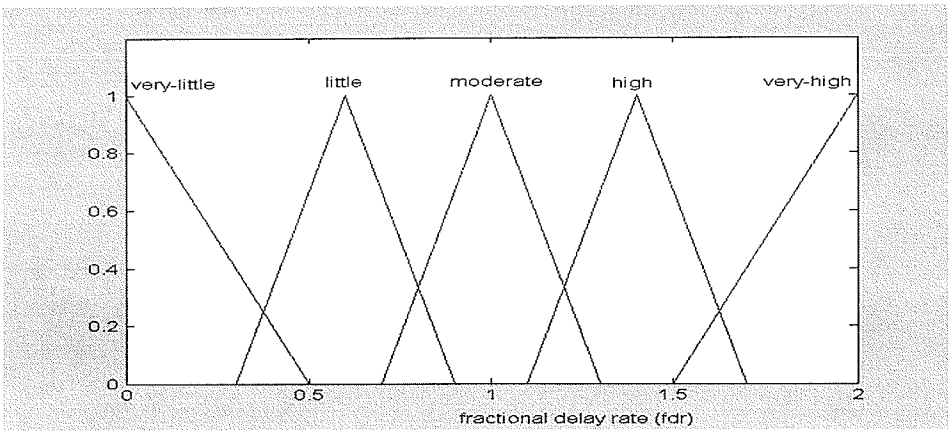


Figure 4.12 Membership functions for *psr, cpsr*, and *fdr*, respectively

After these, a fuzzy rule base, based on the above linguistic description of the main input performance parameters, is developed. Table 4.2 gives the fuzzy control rules designed for our flow control.

Table 4.2. Fuzzy control rules

Rule	*psr*	*cpsr*	*fdr*
1	low	*	very_high
2	moderate	decreasing_fast	very_high
3	moderate	decreasing_slow	high
4	moderate	zero	moderate
5	moderate	increasing_slow	little
6	moderate	increasing_fast	very_little
7	high	decreasing_fast	moderate
8	high	decreasing_slow	little
9	high	zero	very_little
10	high	increasing_slow	very_little
11	high	increasing_fast	very_little

Many inference methods can be applied in the inference engine [87-88]. We have used *Larsen*'s max-product inference method in our design. The widely used *center-of-gravity* method has also been selected as our defuzzification strategy.

4.4.2 Experimental Measurements

In the following, we use UFTPfuzzy and UFTP to represent the protocols with the fuzzy flow controller enabled and disabled, respectively. To evaluate the performance of UFTP with flow control using the designed fuzzy controller and compare it with FTP, the following tests were conducted:

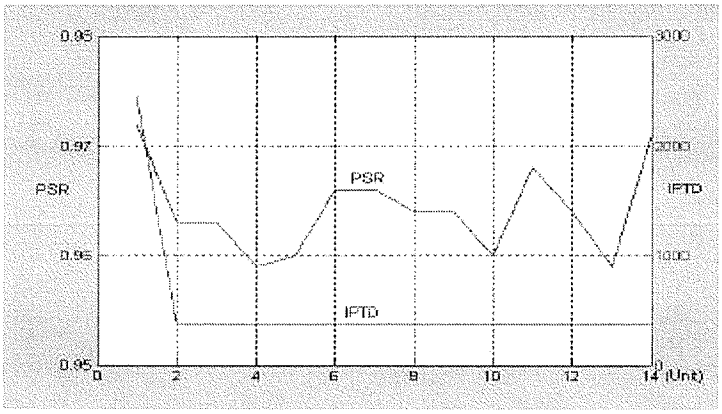
- Tests to monitor transient response of UFTPfuzzy
- Performance tests under moderate traffic scenario
- Performance tests under heavy traffic scenario

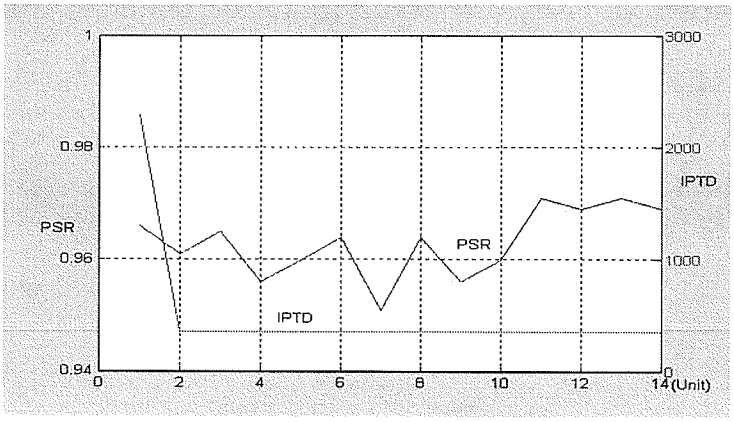
Like in Section 4.3.2, the moderate traffic conditions are weekdays during school hours (9:00am – 5:00pm) and the heavy traffic conditions are generated by running FTP and UFTP at the same time.

One machine (SunOs 5.8, Server) in University of Victoria and another machine (SunOs 5.7, Client) in Lakehead University were used to conduct all the tests. The two machines are connected via high-speed Internet connections. The three files with 15MB, 30MB and 100MB used in conducting the performance tests in Section 4.3.2 were also used in the tests for performance comparison illustrated here.

A. Transient performance monitoring of UFTPfuzzy

Figure 4.13 is the sample transient responses of UFTPfuzzy when downloading the 30MB file. It is observed that the *psr* is high and stable (around 0.965), which indicates that no or minimal congestion occurred during file transferring. It is also observed that *IPTD* is adjusted to its optimal value very fast, only after the 1st unit of packets is transferred.





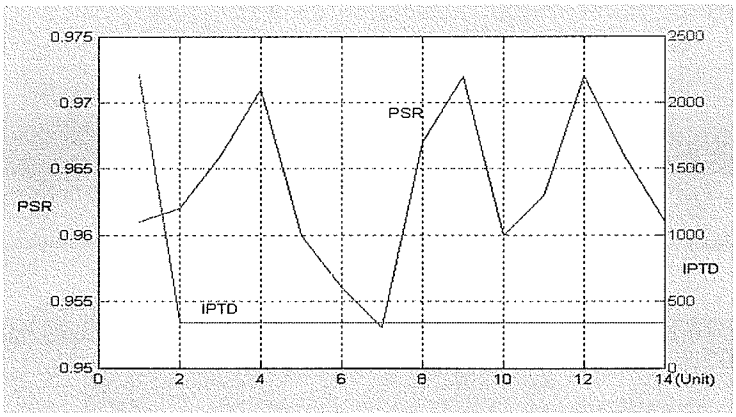


Figure 4.13 Transient performances of UFTPfuzzy, where *IPTD* are measured in μs

B. Performance tests under moderate traffic scenario

Figure 4.14, Figure 4.15, and Figure 4.16 are the measurements of transfer times for downloading the three different-size files under moderate traffic conditions. The measurements for each file are repeated 30 times. From the test results, we see that UFTPfuzzy transfer time is approximately 1.25 times faster than FTP. Also from the test results, we see that UFTPfuzzy transfer time is approximately 1.19 times faster than UFTP when downloading the 15MB file, 1.21 times faster than UFTP when downloading the 30MB file, and 1.22 times faster than UFTP when downloading the 100MB file, which indicates that the fuzzy flow controller works as anticipated.

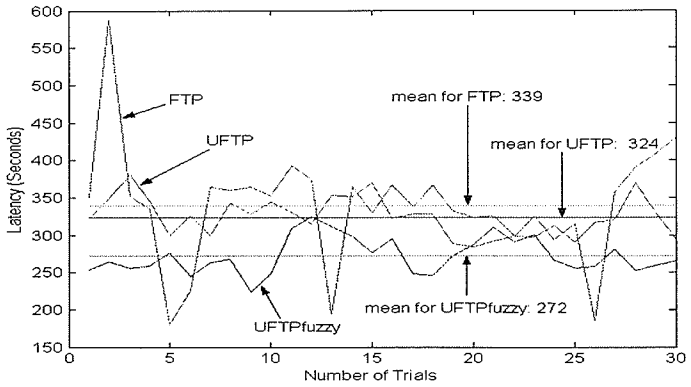


Figure 4.14 Downloading a 15MB file with fuzzy controller under moderate traffic scenario

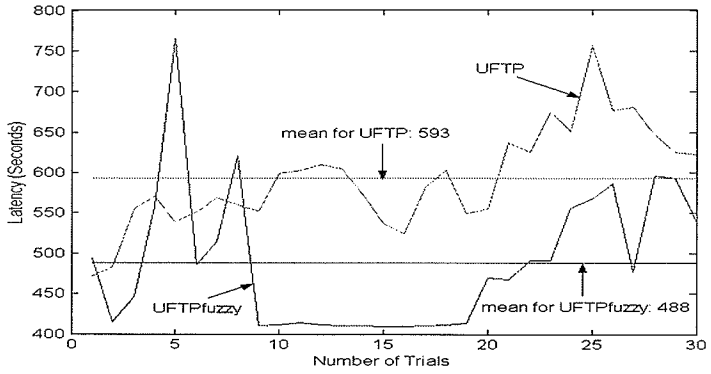


Figure 4.15 Downloading a 30MB file with fuzzy controller under moderate traffic scenario

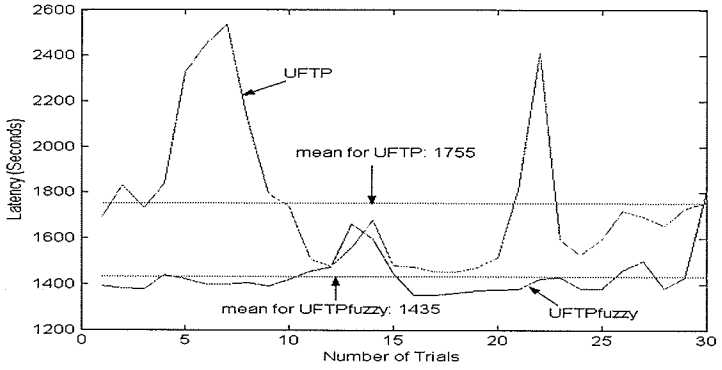


Figure 4.16 Downloading a 100MB file with fuzzy controller under moderate traffic scenario

C. Performance tests under heavy traffic scenario

Figure 4.17, Figure 4.18, and Figure 4.19 are the measurements of transfer times for downloading the three different-size files under heavy traffic conditions. The measurements for each file are also repeated 30 times.

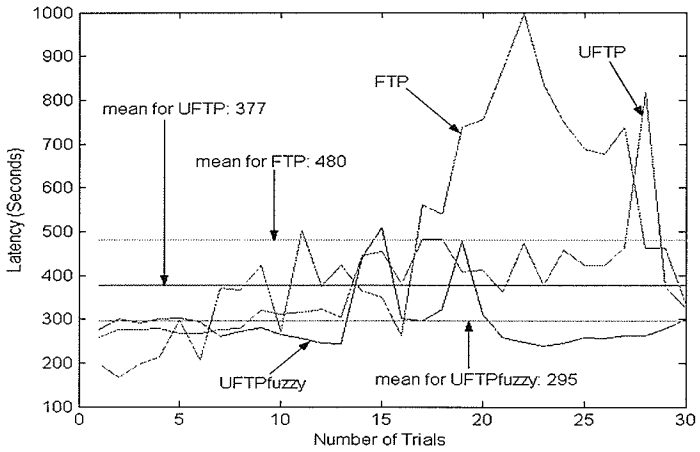


Figure 4.17 Downloading a 15MB file with fuzzy controller under heavy traffic scenario

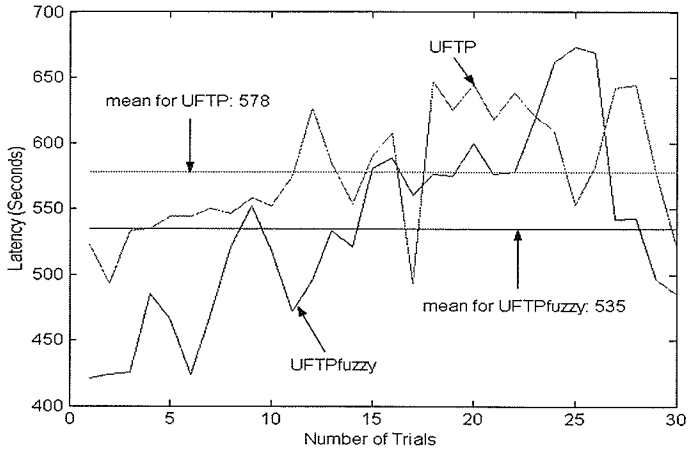


Figure 4.18 Downloading a 30MB File with fuzzy controller under heavy traffic scenario

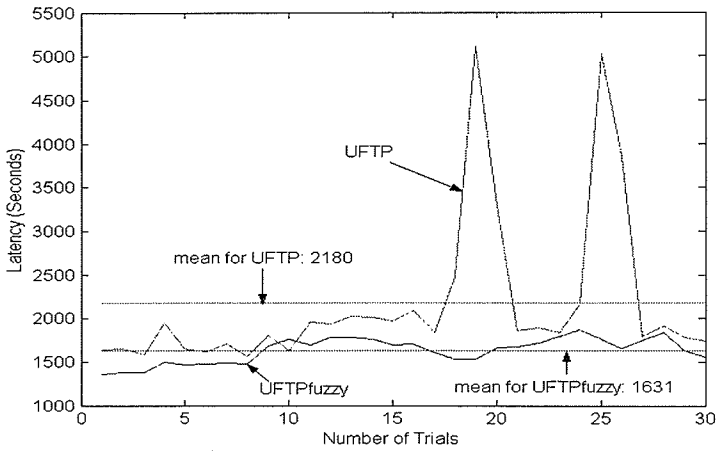


Figure 4.19 Downloading a 100MB file with fuzzy controller under heavy traffic scenario

From the test results, we see that UFTPfuzzy transfer time is approximately 1.63 times faster than FTP. Also from the test results, we see that UFTPfuzzy transfer time is approximately 1.28 times faster than UFTP when downloading the 15MB file, 1.08 times faster than UFTP when downloading the 30MB file, and 1.34 times faster than UFTP when downloading the 100MB file, which indicates that the fuzzy flow controller works as anticipated.

These test results also indicate that when transferring a file using UFTPfuzzy (under both moderate and heavy traffic conditions), the larger the file to be transferred is, the more significant the file transfer throughput improvements are. They also indicate that UFTPfuzzy throughput improvements as compared to FTP are more dramatic under heavy network traffic conditions.

4.5 Summary

In this chapter, we proposed a UDP-based application-level file transfer protocol, UFTP, to improve large-set data transport over the Internet. UFTP is designed for moving "big" files across large wide-area networks (WANs) where traditional file transfer protocols are found to be very inefficient. Compared to FTP, which uses TCP as its transport-level protocol, UFTP uses UDP packets to send data. A "credit-granted" acknowledgement scheme for data acknowledgements is also introduced but not discussed. A dynamic control algorithm based on rough sets or fuzzy logic is applied to UFTP flow control to accomplish a real time data-transferring rate adjustment task. The operations of UFTP are explained and its performance is tested under both moderate and heavy traffic conditions.

The experimental results indicate that when transferring a file using UFTP, the larger the file to be transferred is, the more significant the file transfer throughput improvements are. They also indicate that UFTP throughput improvements as compared to FTP are more dramatic under heavy network traffic conditions. From the experimental results, it can be seen that the UFTP protocol is up to 1.63 times faster than FTP and thus a considerable reduction in network latency is achieved.

Chapter 5

Improved Data Transport Using ALRO

5.1 Motivation for Designing an Application Layer Routing Options (ALRO) scheme

The basic file transfer process in place to date consists of a client and a server. The client requests a file and the server sends the file to the client. In the most basic scenario the file is divided into packets at the server and these packets are delivered to the client over a connection established through the Internet using a transport layer protocol known as TCP. The convenience of TCP is that the end applications basically read and write data to and from streams, and TCP takes care of the network and communication details such as flow, congestion and error control.

The current state of the Internet has routers providing the connectivity between hosts on the Internet. Routing tables are used to determine best paths as specified by a number of factors including shortest path, available bandwidth, and policies and agreements between different carriers. Routes do not tend to change during sessions with congestion being avoided by mechanisms within the transport layer protocol TCP. That is if a server does not receive information from the client it may infer congestion and effectively throttle back its transmission. This "back off" mechanism was designed for traffic flow

and congestion control, but on today's much higher-speed and more reliable Internet network resources will not be fully utilized when this mechanism is in use. This mechanism may increase network latency, and decrease greatly the network performance.

To tackle this problem, an *Application Layer Routing Options (ALRO)* mechanism is proposed in this chapter. The basic idea is to reroute the flow of packets along an alternative route once congestion is inferred to maintain the data transferring speed. Currently there are no examples for this in any well-known protocols. Mechanisms do exist for supporting this type of function but applications have not been developed to exploit this type of functionality.

The objective for this mechanism is to investigate ways in which the network and its topology itself can be more efficiently utilized as part of an end-to-end performance gain.

5.2 Design of the ALRO Scheme

The ALRO mechanism is implemented through the UDP-based application level file transfer protocol, UFTP, with the rough/fuzzy controller disabled in order to avoid the use of the "back off" mechanism. Figure 5.1 shows the basic idea of this mechanism. It works as follows within a data transfer session. A data connection is established (forward channel) between a UFTP server and a UFTP client. During the transfer a router along the path becomes congested, effectively throttling back the server. The server makes a

soft decision based on control information received from the client. The server then decides to transfer data over an alternative route. The alternative route is configured as a prototype function implemented at a known host dedicated to redirecting the data stream.

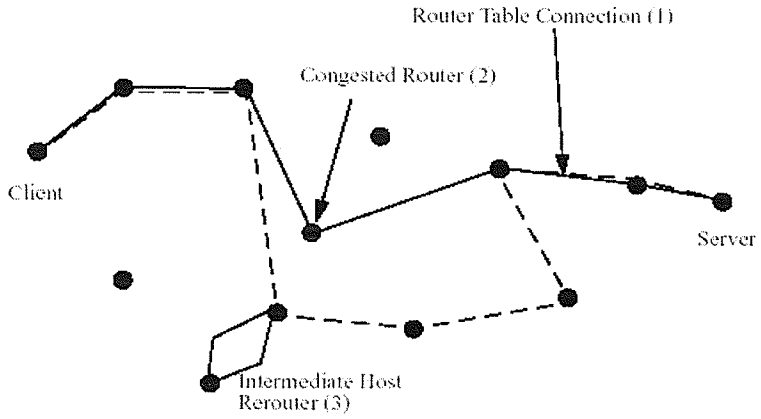


Figure 5.1. Basic idea of ALRO scheme

The detail operations involved in an ALRO-UFTP file transmission are as follows:

1. The UFTP client opens up a TCP connection with the UFTP server for exchanging commands.
2. The packet size is set by the client and forwarded to the server.
3. The client requests a file listing over the TCP connection. The server returns a list of available files and their respective sizes. The client chooses a file from the list.
4. The client/server start the transmission time test to calculate the inter-packet transmission delay.

5. Based on the size of the file to be transferred, the client determines the number of "sections" that the file is divided into. Each section is further divided into many "units". For example, a "section" might be 10Mbyte, and a "unit" 2Mbyte.
6. The client issues a "SEND" command to the server to transfer the first "unit" of the file to be retrieved, taking note of the file's size from the previously acquired file listing.
7. The server opens a UDP connection with the client and starts sending the first "unit" of the file as a stream of UFTP packets, separated by the inter-packet transmission delay (step 3) such that one packet is transmitted for each specified delay interval. The UFTP header of each packet contains the file name and a sequence number to indicate the position of the data in the file.
8. The client stores the incoming UFTP packets and keeps track of the ones it received until a timeout occurs.
9. The client issues a "NWRT" command to the server to reroute packets if congestion inferred (based on packet loss rate).
10. The server sends the next "unit" without doing retransmission of the missing/corrupted packets in the previous unit(s). The traffic route might be changed if the "NWRT" command issued in Step 9.
11. Step 8, 9 and 10 is repeated until the last "unit" of this "section" is sent out.

12. The client generates a list of missing/corrupted packets within this "section" and sends them, over the TCP connection, in a "retransmission" request to the server.
13. The server retransmits the requested UFTP packets in the same format as the original transmission. The client stores the incoming UFTP packets and keeps track of the ones it received until a timeout occurs.
14. The client issues a "NWRT" command to the server to reroute packets if congestion inferred (also based on packet loss rate).
15. This retransmission continues until all of the packets of this "section" are completely received by the client.
16. Then the client issues a "SEND" command to the server to send the first "unit" of the next "section". The server sends the requested "unit".
17. Step 8-16 is repeated until the last "section" of the file is completely received.
18. The client extracts the original data from the received packets and writes the transferred file to disk. The whole file transfer is then complete.

5.3 Experimental Results

To evaluate the performance of ALRO scheme, the following tests were conducted:

- Tests of the ALRO performance using different "packet drop rate" as the congestion threshold

- Performance tests to compare ALRO-UFTP with UFTPfuzzy & FTP

To simplify our experiments, only one "relay server" is used to test our ALRO mechanism in this study. The following three machines were used to conduct the tests: one remote machine *galois* (Server) in Victoria, BC, another remote machine *giant* (Client) in Lakehead University, and a local machine *galliano* (Relay server) in University of Manitoba. These three machines are connected to each other via high-speed Internet connections. A large-size file with 30MB (30,093,344 bytes) was used in each test.

Using traceroute, we found that the route from *galois* directly to *giant* is quite different from the route from *galois* via *galliano* to *giant*.

We have chosen the packet drop rate (*pdr*), i.e., the packet transferring decrease rate, as the traffic congestion parameter. Congestion is assumed if packets are dropped dramatically compared to the previous transferring.

To better see the performance of ALRO scheme, we have chosen the relatively busy Internet time (weekday afternoons) to conduct our experiments.

A. Tests of the ALRO performance using different packet drop rate as the congestion threshold

All the experimental results are the averages of three tests around the same time.

The test results are shown in Table 5.1, where

CTValue: congestion threshold value

RSTimes: route switch times

TTTimes: total transmission/retransmission times

TVRate: traffic volume rate, i.e., percentage of the number of packets sent along the alternative route among the total packets transferred

TFTime: total file transfer time

Table 5.1. ALRO vs Congestion Threshold

CTValue	*pdr* = 2%	*pdr* = 5%	*pdr* = 10%	*pdr* = 15%
RSTimes	2	0.7	1.3	0
TTTimes	21	21	32	24
TVRate	50%	64%	53%	0
TFTime	144	163	146	154

From this table, we observe the following:

- *RSTimes* are very small (compared to *TTTimes*), which means the network is very stable with congestion seldom happening on the route.
- No congestion is inferred when *CTValue* set to 15% (or higher)
- *TVRate* values show that the network traffic is very balanced when congestion is inferred.

- *TFTime* values are very close to each other (which make sense as the number of times the route switched is quite small).

Based on this observation, we set *CTValue* to 10% to infer traffic congestion in the following tests.

B. Performance tests to compare ALRO-UFTP with UFTPfuzzy & FTP

Figure 5.2 shows the average transfer times during busy time (weekday afternoons) for ALRO-UFTP, UFTPfuzzy & FTP measured in *seconds*. The measurements were repeated 15 times. From the test results, we see that ALRO-UFTP transfer time is 1.38 times faster than UFTPfuzzy and 3 times faster than FTP, which illustrates that ALRO-UFTP is efficient in data transfer and works to balance the load on the network(s).

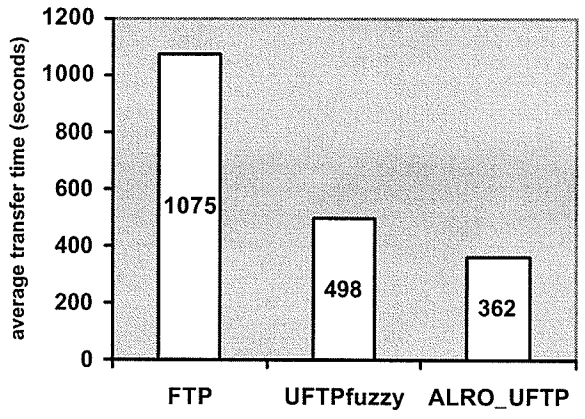


Figure 5.2 Results of ALRO-UFTP, UFTPfuzzy & FTP tests

5.4 Summary

The basic file transfer architecture is based on the conventional client server paradigm with the Internet providing the interconnect infrastructure. In order to avoid/control traffic congestion, the most common scenario is that the server effectively throttles back its transmission when congestion is inferred. In this chapter, an alternative mechanism, *Application Layer Routing Option* (ALRO), is proposed in order to improve the transport of large-size files over the Internet. In our approach, the server does not throttle back its transmission when congestion is inferred; instead, it will reroute the flow of packets along an alternative route. This alternative route is constructed through a "relay router or server", which is a prototype function implemented at a known host dedicated to redirecting the data stream. Of course as a secondary measure the server should be forced to throttle back if the packet reception rate continues to decrease even though an alternative route is used. The proposed mechanism is implemented through a UDP-based application level file transfer protocol and the performance is tested on the Internet through experiments using only one relay server. The underlying file transfer protocol denoted by UFTP already improves latency on the Internet with improvements often on the order of 1.63 times over the traditional file transfer protocol FTP. The experimental results presented here show that ALRO combined with UFTP attempts to balance the network load when congestion inferred while still obtaining high-speed file transferring.

Chapter 6

Conclusions

The experiments conducted and the resulting data presented have demonstrated the major effect of the UFTP protocol on the latency over the Internet. They have also demonstrated the performance gain when a dynamic flow controller based on fuzzy logic or rough sets is applied. The following conclusions can be made from the experimental results:

- The UFTP protocol is up to 1.63 times faster than the conventional file transfer protocol FTP.

- When transferring a file using UFTP, the larger the file to be transferred is, the more significant the file transfer throughput improvements are.

- UFTP throughput improvements, as compared to FTP, are more dramatic under heavy network traffic conditions.

The main point here is that a controller based on rough sets or fuzzy logic demonstrated an improvement over more traditional methods in transferring large files across the Internet. It should be noted that the concepts illustrated here and their efficiency are not

bound to the underlying transport layer protocol. The rough set or fuzzy logic controller could be applied to improving transfer of data over TCP as well. However, difficulties in developing a prototype would be associated with user level control over TCP parameters which are not easily accessible. In future TCP releases a controller could be built to modify the TCP MIB thereby providing control information without requiring kernel level programming skills.

Besides UFTP, ALRO is another mechanism that was proposed in this thesis in order to improve the transport of large-size files over the Internet. Unlike the traditional congestion control scenario that the server effectively throttles back its transmission when congestion is inferred, ALRO does not throttle back its transmission (through the server) when congestion is inferred; instead, it will reroute the flow of packets along an alternative route. The experimental results presented show that ALRO combined with UFTP attempts to balance the network load when congestion inferred while still obtaining high-speed file transferring. The idea here is to get maximum benefit from underlying protocols by making routing type decisions at the application layer. This work also demonstrates that routing algorithms and consequently routing themselves can be made more efficient if second shortest path routes were also available in router look-up tables, thereby allowing packets shunting to second best routes if congestion inferred.

Future research may focus on further evaluating and optimizing the performance of ALRO-UFTP. The following tasks are then highly recommended for future considerations:

- Continue the rough sets/fuzzy logic controller studies using Phatpackets as the transport layer [89]. Phatpackets is an extension to the FSTP protocol. It combines multiple servers and multiple connections with estimates of file sizes to download made from bandwidth delay products and flow control made in real time.

- Conduct experiments with ALRO across global WANs with internets connected via a number of disparate speed connections, where real traffic congestion occurs more often and a number of different alternative routes can be selected. It is expected that our new mechanism will work much better under such a global WAN. In that case, a proper value can possibly be set for the packet loss rate (to infer congestions) to reflect the real traffic congestion.

- Extend the basic scheme of ALRO for selecting multiple relay servers. An extension to the basic scheme would be in developing the application to utilize alternative routes via many different relay servers to varying degrees during the session. At this stage, a neural or rough neural network could be designed for selecting the best routes.

- The ALRO option should be transparent to the user and if sufficient efficacy can be demonstrated, implemented more directly within routers or edge devices.

The problem of alternative route selection albeit important is not addressed in this thesis. The results of the research however can be used to illustrate the benefits of multiple path routing which could be implemented in actual routers if Internet routing algorithms provided second shortest paths in addition to shortest paths information.

References

1. V. N. Padmanabhan, and Jeffery C. Mogul, "Improving HTTP Latency", Proceedings of the Second International World Wide Web Conference, Chicago, IL, pages 995-1005, Oct. 1994

2. R. Fielding, J. Gettys, J. Mogul, H. Nielsen, and T. Berner-Lee, "Hypertext Transfer Protocol-HTTP/1.1", IEFT, RFC 2068, Jan. 1997

3. B.S. Noghani, S. Kretschmann, and R.D. McLeod, "Reducing Latency on the Internet using 'Component-Based Download' and 'File-Segment Transfer Protocol'", PDPTA '2001 – Las Vegas, June 2001

4. http://www.web100.org/rprojects/ - "Automatic TCP Window Tuning and Applications"

5. M. Crovella and P. Barford, "The Network effects of Pre-fetching", Proceedings of IEEE Infocom'98, San Francisco, CA, 1998

6. A.N. Eden, B.W. Joh, and T. Mudge, "Web latency reduction via client-side prefetching", 2000 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS (Cat. No.00EX422), pp. 193-200, Austin, USA, April 2000

7. D. Foygel and D. Strelow, "Reducing Web latency with hierarchical cache-based prefetching", Proceedings of 2000 International Workshop on Parallel Processing, pp. 103-108, Toronto, Ont., Canada, Aug. 2000

8. V.N. Padmanabhan, J.C. Mogul, "Using predictive prefetching to improve World Wide Web latency", Computer Communication Review, Vol. 26, No. 3, pp. 22-36, July 1996

9. E. Cohen and H. Kaplan, "Prefetching the means for document transfer: a new approach for reducing Web latency", Proceedings of IEEE INFOCOM 2000, pp. 854-863, Tel Aviv, Israel, March 2000

10. J. Leis, "Rapid display of Web content: a simple method for prefetching Web files", Computing & Control Engineering Journal, vol.13, no.3, pp. 149-152, June 2002

11. R.P. Klemm, "WebCompanion: a friendly client-side Web prefetching agent", IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 4, pp. 577-594, 1999

12. A. J. Smith, "Cache memories", ACM Computing Surveys, vol. 14, pp. 473-530, Sept. 1982

13. S. Williams, M. Abrams, C.R. Standridge, "Removal policies in network caches for world-wide web documents", Proceedings of ACM SIGCOMM'96, pp. 293-305, July 1996

14. G.P. Chandranmenon and G. Varghese, "Reducing Web latency using reference point caching", Proceedings of IEEE INFOCOM 2001, pp. 1607-1616, Anchorage, AK, USA, April 2001

15. X. Tang and S.T. Chanson, "Coordinated en-route Web caching", IEEE Transactions on Computers, vol. 51, no. 6, pp. 595-607, June 2002

16. G.P. Chandranmenon, "Reducing Internet latency using precomputed hints", Ph.D. Thesis, WASHINGTON UNIVERSITY, 1999

17. P. Rodriguez, A. Kirpal, and E.W. Biersack, "Parallel-access for mirror sites in the Internet", Proceedings of IEEE INFOCOM 2000, pp. 863-873, Tel Aviv, Israel, March 2000

18. J. Gutyon and M. Schwartz, "Locating Nearby Copies of Replicated Internet Servers", Proceedings of SIGCOMM'95, Aug. 1995

19. Z. Fei, S. Bhattacharjee, E. W. Zegura and M. H. Ammar, "A Novel Server Selection Technique for Improving the Response Time of a Replicated Service", Proceedings of IEEE INFOCOM 1998

20. T. Loukopoulos, I. Ahmad, and D. Papadias, "An overview of data replication on the Internet", Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks. I-SPAN'02, pp. 31-36, Makati City, Metro Manila, Philippines, May 2002

21. A. Leon-Garcia and I. Widjaja, "Communication Networks: Fundamental Concepts and Key Architectures", The McGraw-Hill Companies, 2000

22. D. Sanghi, A.K. Agrawala, O. Gudmundsson, and B. Jain, "Experimental Assessment of End-to-End Behavior on Internet", Proceedings of IEEE Infocom'93, San Francisco, CA, March 1993

23. Partho P. Mishra, Dheeraj Sanghi, and Satish K. Tripathi, "TCP Flow Control in Lossy Networks: Analysis and Enhancement", IFIP Trans. C-13 Computer

Networks, Architecture and Applications, pages 181--193, 1993. Elsevier North Holland

24. A. Mankin and K. Thompson, "Limiting Factors in the Performance of the Slow-Start TCP Algorithms", Proceedings of USENIX Winter Conference'89, pp. 219-228, San Diego, California, Jan. 1989

25. J. Zhang and R.D. McLeod, "A UDP-Based File Transfer Protocol with Flow Control using Fuzzy Logic Approach", Proc. of the IEEE CCECE'03, Montreal, Canada, May 2003

26. J. Zhang and R.D. McLeod, "Application Layer Routing Options for Efficient Data Transport Over The Internet", Proceedings of the 2002 IEEE Canadian Conference on Electrical & Computer Engineering, Winnipeg, Manitoba, Canada, May 2002

27. F. Rosenblatt, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms", Spartan Press, 1961

28. M. Minsky and S. Pappert, "Perceptrons: An Introduction to Computational Geometry", MIT Press, Cambridge, MA, 1969

29. J.M. Zurada, "Introduction to Artificial Neural Systems", West Publishing Company, 1992

30. I. Rechenberg, "Cybernetic Solution Path of an Experimental Problem", Ministry of Aviation, Royal Aircraft Establishment (U.K.), 1965

31. J.H. Holland, "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1975 (second edition by: MIT Press, 1992)

32. L.J. Fogel, A.J. Owens, and M.J. Walsh, "Artificial Intelligence through Simulated Evolution", Wiley, 1966

33. D.E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, 1989

34. L. A. Zadeh, "Fuzzy Sets", Information and Control, 1965, Vol. 8, pp. 338-353

35. E.H. Mamdani, "Applications of Fuzzy Algorithms for Simple Dynamic Plant", Proc. IEE, 121, pp. 1585-1588, 1974

36. R. Cheng and C. Chang, "Design of a Fuzzy Traffic Controller for ATM Networks", IEEE/ACM Trans. Networking, vol. 4, no. 3, pp. 337-344, June 1996

37. M. Collett and W. Pedrycz, "Application of Neural Networks for Routing in Telecommunications Networks", Proc. IEEE GLOBECOM'93, pp. 1001-1006, 1993

38. L. A. Zadeh, "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes", IEEE Trans. On Systems, Man, and Cybernetics, 2:28-44, 1973

39. D. Marr, "Vision", W.H. Freeman, San Francisco, CA, 1982

40. Z. Pawlak, "Rough sets", International Journal of Computer and Information Sciences, vol. 11, pp. 341-356, 1982

41. Z. Pawlak, "Rough sets: theoretical aspects of reasoning about data", Kluwer Academic Publishers, Dordrecht, 1991

42. Z. Pawlak, "Rough sets: present state and future prospects", ICS Research Report **32/95**, Institute of Computer Science, Warsaw Institute of Technology, 1995

43. A. Skowron, C. Rauszer, "The Discernibility Matrices and Functions in Information Systems", In Slowinski, R. (Ed.), Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory, Kluwer Academic Publishers, Dordrecht, 1992, 331-362.

44. A..Skowron, R.W. Swiniarski, "Information granulation and pattern recognition", In: S. Pal, L. Polkowski, A. Skowron (Eds.), Rough-Neuro Computing. Physica-Verlag, Berlin, 2002, 636-670.

45. A.R. Bonde and S. Ghosh, "A Comparative Study of Fuzzy Versus 'Fixed' Thresholds for Robust Queue Management in Cell-Switching Networks", IEEE/ACM Trans. Networking, vol. 2, no. 4, pp. 337-344, August 1994

46. Q. Razouqi, S. Joo, and S. Ghosh, "Performance Analysis of Fuzzy Thresholding-Based Buffer Management for a Large-Scale Cell-Switching Network", IEEE Trans. Fuzzy Systems, vol. 8, no. 4, pp. 425-441, August 2000

47. G. Ascia, V. Catania, G. Ficili, and D. Panno, "A Fuzzy Buffer Management Scheme for ATM and IP Networks", Proceedings of IEEE INFOCOM'2001, pp. 1539-1547, 2001

48. K. Uehara and K. Hirota, "Fuzzy Connection Admission Control for ATM Networks Based on Possibility Distribution of Cell Loss Ratio", IEEE Journal

on Selected Areas in Communications, vol. 15, no. 2, pp. 179-190, February 1997

49. Q. Ren and G. Ramamurthy, "A Real-Time Dynamic Connection Admission Controller Based On Traffic Modeling, Measurement, and Fuzzy Logic Control", IEEE Journal on Selected Areas in Communications, vol. 18, no. 2, pp. 184-196, February 2000

50. G. Ascia, V. Catania, G. Ficili, S. Palazzo, and D. Panno, "A VLSI Fuzzy Expert System for Real-Time Traffic Control in ATM Networks", IEEE Trans. Fuzzy Systems, vol. 5, no. 1, pp. 20-31, Feb. 1997

51. H. Lim and B. Qiu, "Fuzzy Logic Target Utilization and Prediction for Traffic Control", Proc. IEEE GLOBECOM'2000, pp. 1644-1648, 2000

52. A. Pitsillides, Y. Sekercioglu, and G. Ramamurthy, "Effective Control of Traffic Flow in ATM Networks Using Fuzzy Explicit Rate Marking (FERM)", IEEE Journal on Selected Areas in Communications, vol. 15, no. 2, pp. 209-225, February 1997

53. S. Kuan and L. Andrew, "Performance of Fuzzy Logic ABR Rate Control with Large Round Trip Times", Proc. IEEE GLOBECOM'98, 1998

54. R. Hu and D. Petr, "A Predictive Self-Tuning Fuzzy-Logic Feedback Rate Controller", IEEE/ACM Trans. Networking, vol. 8, no. 6, pp. 697-709, Dec. 2000

55. A. Pitsillides, Y. Sekercioglu, and G. Ramamurthy, "Fuzzy Backward Congestion Notification (FBCN) Congestion Control in Asynchronous Transfer Mode (ATM) Networks", Proc. IEEE GLOBECOM'95, pp. 280-285, 1995

56. B. Qiu, "A Predictive Fuzzy Logic Congestion Avoidance Scheme", Proc. IEEE GLOBECOM'97, pp. 967-971, 1997

57. B. Chen, S. Peng, and K. Wang, "Traffic Modeling, Prediction, and Congestion Control for High-Speed Networks: A Fuzzy AR Approach", IEEE Trans. Fuzzy Systems, vol. 8, no. 5, pp. 491-508, Oct. 2000

58. P. Chemouil, J. Khalfet, and M. Lebourges, "A Fuzzy Control Approach for Adaptive Traffic Routing", IEEE Commun. Mag., pp. 70-76, July 1995

59. A.V. Vasilakos, K.G. Anagnostakis, and W. Pedrycz, "Application of Computational Intelligence Techniques in Active Networks", Soft Computing, vol.5, no.4, pp. 264-271, Aug. 2001

60. V. Catania, G. Ficili, S. Palazzo, and D. Panno, "A Comparative Analysis of Fuzzy Versus Conventional Policing Mechanisms for ATM Networks", IEEE/ACM Trans. Networking, vol. 4, no. 3, pp. 449-459, June 1996

61. D. Slonowsky, Q. Jiang, R. Srinivasan, "A Fuzzy Dynamic Bandwidth Re-Allocator", Proceedings of the 2002 IEEE Canadian Conference on Electrical & Computer Engineering, Winnipeg, Manitoba, Canada, May 2002

62. Q. Jiang,R. Srinivasan, D. Slonowsky, "Measurment Based Traffic Prediction Using Fuzzy Logic", Proceedings of the 2002 IEEE Canadian Conference on Electrical & Computer Engineering, Winnipeg, Manitoba, Canada, May 2002

63. D. Jensen, "B-ISDN Network Management by a Fuzzy Logic Controller", Proc. IEEE GLOBECOM'94, pp. 799-804, 1994

64. G. Edwards and R. Sankar, "Hand-Off Using Fuzzy Logic", Proc. IEEE GLOBECOM'95, pp. 524-528, 1995

65. G. Edwards and R. Sankar, "A Predictive Fuzzy Algorithm for High Performance Microcellular Handoff", Proc. IEEE GLOBECOM'97, pp. 987-990, 1997

66. S. Lau, K. Cheung, and J. Chuang, "Fuzzy Logic Adaptive Handoff Algorithm", Proc. IEEE GLOBECOM'95, pp. 509-513, 1995

67. K.W. Shum and C. Sung, "Fuzzy Layer Selection Method In Hierarchical Cellular Systems", Proc. IEEE GLOBECOM'96, pp. 1049-1053, 1996

68. V.I. Levin, "An Analysis of Computer Networks with Nondeterminate Parameters using Nondeterministic Logic", Automat. Contr. Comput. Sci., vol. 25, no. 5, pp. 17-24, 1991

69. J. Postel and J. Reynolds, "File Transfer Protocol", IETF, RFC 959, ISI, Oct. 1985

70. B.S. Noghani, S. Kretschmann, and R.D. McLeod, "Reducing Latency on the Internet using 'Component-Based Download' and 'File-Segment Transfer Protocol'", Proceedings of PDPTA'01 – Las Vegas, USA, June 2001

71. Information Sciences Institute, "Transmission Control Protocol", IETF, RFC 793, Sep. 1981

72. A. Karn, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", Proc. ACM SIGCOMM'87, 1987

73. V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance", IETF, RFC 1323, May 1992

74. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options", IETF, RFC 2018, Sun Microsystems, Oct. 1996

75. J.C. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", Proc. ACM SIGCOMM'96, 1996

76. K. Sollins, "The TFTP Protocol (Revision 2)", IETF, RFC 1350, MIT, July 1992

77. J. Poster, "User Datagram Protocol", IETF, RFC 768, USC/ISI, Aug. 1980

78. A. Mrozek, "Rough Sets in Computer Implementation of Rule-Based Control of Industrial Processes". In: R. Slowinski (Ed.), Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory. Dordrecht: Kluwer Academic Publishers, 1992, 19-31.

79. T. Munakata, "Rough Control: Basic Ideas and Applications". In: P.P. Wang (Ed.), Second Annual Joint Conf. on Information Sciences (JCIS'95), Wrightsville Beach, North Carolina, 28 Sept.-1 Oct. 1995, 1135-1139.

80. E. Czogala, A. Mrozek, Z. Pawlak, "The Idea of a Rough Fuzzy Controller and Its Application to The Stabilization of a Pendulum-Car System", Fuzzy Sets and Systems, vol. 72, 1995, 61-73.

81. A. Mrozek, L. Plonka, J. Kedziera, "The Methodology of Rough Controller Synthesis". In: Proc. of the 5th IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'96), New Orleans, Louisiana, 8-11 Sept. 1996, 1135-1139.

82. Z. Pawlak, T. Munakata, "Rough Control: Application of Rough Set Theory to Control". In: Proc. of the 4th European Congress on Intelligent Techniques and Soft Computing (EUFIT'96), Germany, 2-5 Sept. 1996, 209-218.

83. J.F. Peters, K. Ziaei, and S. Ramanna, "Approximate Time Rough Control: Concepts and Application to Satellite Attitude Control". In: L. Polkowski, A. Skowron (Eds.), Lecture Notes in Artificial Intelligence, vol. 1424, 1998, 491-498.

84. J.F. Peters, A. Skowron, and Z. Suraj, "An application of rough set methods in control design", Fundamenta Informaticae, vol. 43, nos. 1-4, 2000, 269-290.

85. http://www.idi.ntnu.no/~aleks/rosetta/ - the *ROSETTA* WWW homepage

86. H.S. Nguyen, S.H. Nguyen, "Discretization methods in data mining", In: L. Polkowski and A. Skowron (Eds.), Rough Sets in Knowledge Discovery. Physica-Verlag, Berlin, 1998, pp. 451-482

87. W. Pedrycz, "Fuzzy control and fuzzy systems", Somerset, England: Research Studies Press, 1993

88. W. Pedrycz, "Fuzzy Sets Engineering", CRC Press, Boca Raton, 1995

89. S. Huang, S. Silverman, and R. D. McLeod, "Phatpackets: Implementation and Experimentation," *submitted to The 3rd International Conference on Internet Computing* , Las Vegas, USA, June 2002, also available as a phatpackets white paper at www.phatpackets.com