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

LIST OF FIGURES

LIST OF TABLES

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.

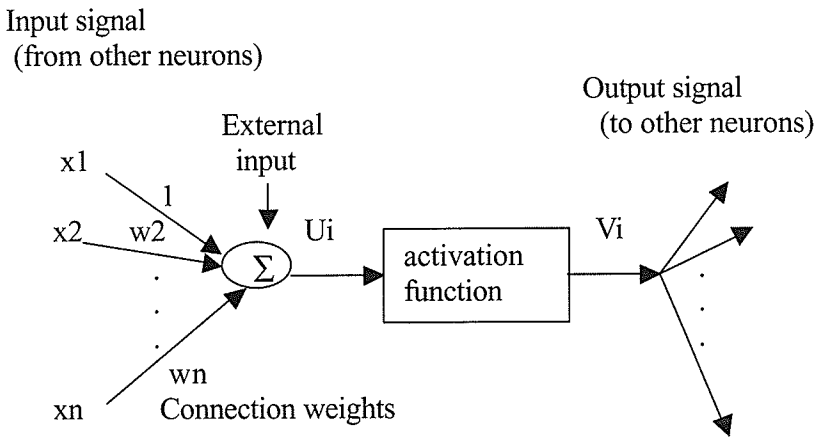

Figure 2.1. Processing element (neuron)

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.

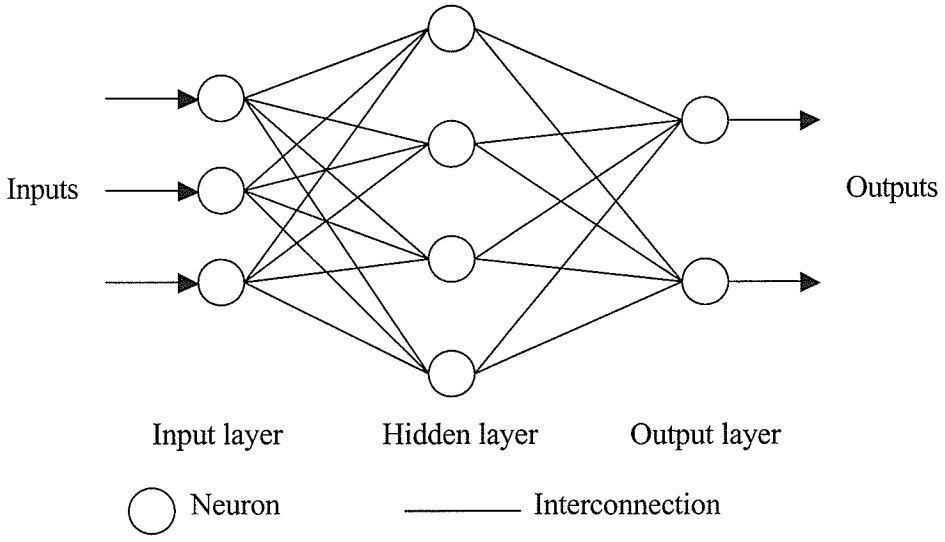

Figure 2.2. Three-layer feed-forward neural network

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.