7-5-2005

A Fuzzy Logic Based Controller to Provide End-To-End Congestion Control for Streaming Media Applications

Bay Pavlick
University of South Florida

A Fuzzy Logic Based Controller to Provide End-To-End Congestion Control for Streaming Media Applications

by

Bay Pavlick

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Miguel A. Labrador, Ph.D.
Kenneth J. Christensen, Ph.D.
Dewey J. Rundus, Ph.D.

Date of Approval:
July 5, 2005

Keywords: fuzzy inference system, congestion collapse, tcp friendliness, udp, voice and video applications

Table of Contents

List of Figures

A Fuzzy Logic Based Controller to Provide End-To-End Congestion Control for Streaming Media Applications

Bay Pavlick

ABSTRACT

The stability of the Internet is at risk if the amount of voice and video traffic continues to increase at the current pace. While current transport layer protocols do work well for most applications, they still present some problems. TCP is reliable, tracks the state of some network conditions and reacts drastically to an indication of congestion. TCP serves data-oriented applications very well but it can lead to unacceptably low quality for streaming applications by multiplicatively reducing the congestion window upon a sign of congestion. The other main transport layer protocol, UDP, provides good service for streaming applications but is not friendly to TCP and can cause the well-known existing congestion collapse problem in the Internet.

This thesis proposes a new protocol to provide a good service for voice and video applications while being friendly to TCP and solving the congestion collapse problem. The protocol utilizes a fuzzy logic controller that considers network related information to govern the application's sending rate while satisfying the user's needs. Using network information such as the available bandwidth, Packet Loss Rates (PLR), and Round Trip Times (RTT) a fuzzy inference system optimizes the application's send rate to meet the requested rate in a smooth manner without wasting network resources unnecessarily.

The fuzzy logic controller is designed and its performance evaluated using MATLAB model simulations. The results indicate that the fuzzy controller solves the congestion collapse problem by reducing the number of undelivered packets into the network by nearly 100%. It provides smooth transition changes as demonstrated by the controlled UDP flow utilizing an estimated 44% more of the available bandwidth to smooth the send rate than the TCP flow in a highly varying bandwidth environment. The controller also remains friendly to TCP which was demonstrated to share the bandwidth at nearly 50% with one other competing controlled UDP flow.

Chapter One

Introduction

1.1 Introduction and Motivations

The Internet is increasingly being used for non-data applications such as voice and streaming video as noted in Girod *et al.* [27]. This trend shows the behavior of network flows changing from mostly short bursty traffic to the inclusion of flows that are time sensitive and of longer duration. The architecture of the Internet then loses some of its effectiveness to provide stability, fairness and quality to the applications as the current transport layer protocols do not work together very well. Transmission Control Protocol (TCP) does act fairly towards other flows and has controls for preventing the application from sending more data than the network can handle. These controls make TCP react drastically to congestion events, providing a fluctuating transmission rate to applications. However, the reliability of the TCP protocol has made it the protocol of choice for data-oriented applications. As such, most applications use the TCP protocol such as file transfers (using File Transfer Protocol), emails (using Simple Mail Transfer Protocol) and web browsing (using Hyper Text Transfer Protocol).

User Datagram Protocol (UDP) does provide applications with a rather steady transfer rate but it is unreliable and lacks any end-to-end flow and congestion control mechanism. As a result, UDP is the protocol of choice for streaming applications since they

can afford some packet loss but are sensitive to small amounts of jitter. This allows the application to receive a smoother rate over time but at TCP's expense, as UDP does not react to congestion. Furthermore, UDP will continue inserting packets into the network regardless of whether or not they are reaching their destination. These are the well-known TCP-friendliness and congestion collapse problems documented in Floyd and Fall [5].

Recently, the inclusion of flow and congestion control mechanisms to end-to-end streaming-oriented protocols has been emphasized as an important measure to deal with these problems. These new mechanisms need to satisfy the following goals:

1. Avoid congestion collapse: In order to solve the congestion collapse from undelivered packets, the application needs to be aware if the network has no (or limited) available bandwidth along the destination path so that it can restrict its flow to what is appropriate. It will need to restrict bandwidth if there is no available bandwidth and thus not insert packets into the network that won't make it to their destination.
2. Smoothness for streaming media applications: The mechanism must also account for the smoothness given to the application. For TCP, upon finding congestion, it will back off multiplicatively. This reaction would make a streaming application jitter. The solution needs to provide a less drastic varying rate.
3. TCP friendly: If the mechanism is to be TCP friendly, it needs to additionally consider how TCP would respond and take only an appropriate amount of bandwidth as guided by the TCP Response function.

This thesis proposes a new transport layer protocol which utilizes a controller that, using network information such as available bandwidth, Packet Loss Rates, and Round Trip Times, meets the goals and reacts appropriately to signs of congestion. The controller uses this information and the TCP response function to determine what TCP would do in the same situation. However, instead of reacting to congestion using the Multiplicative Decrease strategy of TCP, it reacts with an optimized transmission rate based on the network information, the application's requested rate and considering the response of a TCP flow. In addition, the controller considers the history of the rate it has provided in order to smooth the sending rate and minimize drastic changes in available bandwidth. The general inputs and output of the controller is shown in Figure 1.

The controller is based on Fuzzy Logic given the imprecision of the network information utilized and the rather large number of variables involved in the optimization. The fuzzy logic is used to optimize the output (the application send rate) given the inputs (the network information and TCP response) and a set of rules to guide the output. Fuzzy controllers can be designed and evaluated in MATLAB and Simulink and then expressed in the C language to be included in operating systems.

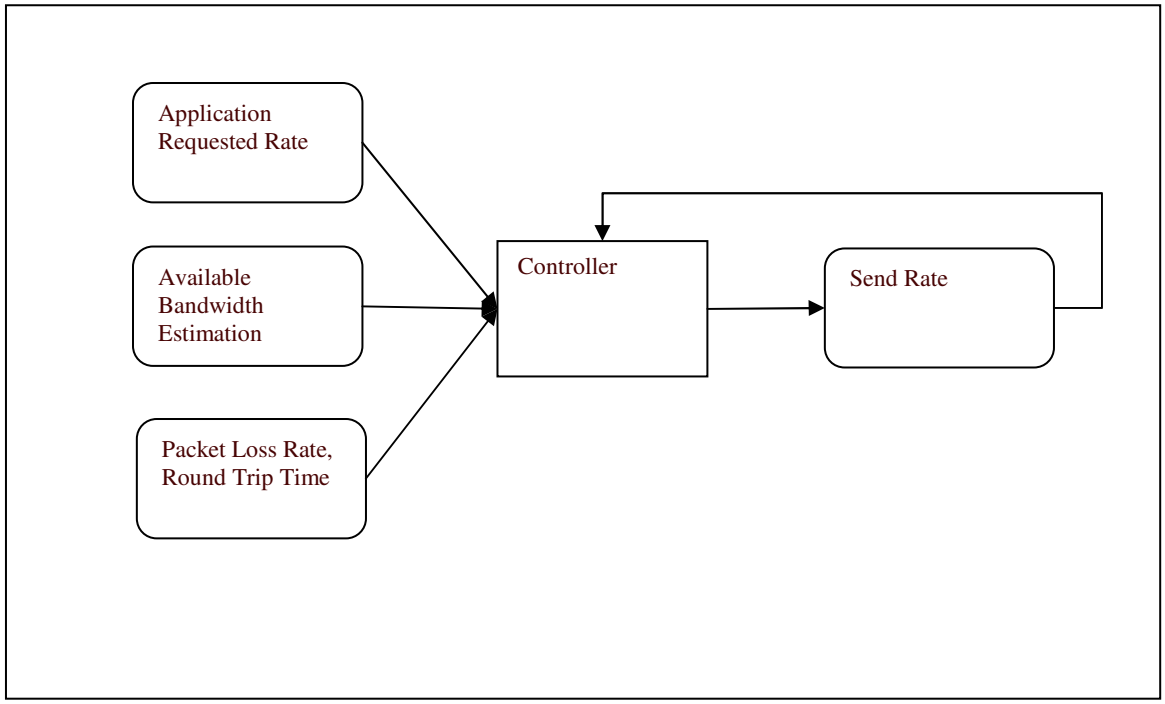

Figure 1 – The General Inputs and Output of the Controller.

1.2 Contributions of the Thesis

This thesis proposes a new transport layer protocol that includes flow and congestion control for streaming applications. The main contributions included in the thesis are:

1. A Fuzzy Inference System (FIS) to implement flow and congestion control at the transport layer for streaming applications. The FIS finds the optimum rate based on the application requirements, information in the network, and the response of TCP under the same conditions.
2. A FIS Controller designed to gather necessary parameters for the FIS, to call the FIS and to return the results to the application. The controller is used as the interface between the application and the FIS.

3. A kernel ready C language implementation of the FIS and FIS Controller. This will allow for testing and analysis under real world conditions.
4. Performance evaluation of the controller system. The system is modeled to show real time results, interactions and effects on other protocol flows.

1.3 Outline of the Thesis

The thesis is organized as follows. Chapter 2 discusses in detail the current Internet transport layer protocols, the characteristics of streaming applications, and previously proposed end-to-end congestion control solutions. Chapter 3 discusses the proposed solution. Chapter 4 details the evaluation methodology and includes the performance evaluation of the controller. Finally, Chapter 5 concludes the thesis and points out directions for future research.

Chapter Two

Literature Review

In this chapter, background information is provided about several concepts and terms in the area of networks, network protocols, measuring network conditions, data and streaming applications and issues transporting information over shared media. In this thesis, available bandwidth estimations, Packet Loss Ratios (PLR) and Round Trip Times (RTT) measurements, fuzzy logic controllers and transport layer protocols are all integrated to solve several important problems such as the TCP-friendliness problem, the congestion collapse problem, and providing streaming media applications with the service they require.

2.1 Available Bandwidth Estimation

In Prasad *et al.* [18], the authors define the terms related to bandwidth estimation and introduce the tools and techniques currently available to measure it. In the paper, *Bandwidth estimation* is a mechanism used to measure network conditions and is accomplished by a variety of tools and involves several other terms. Bandwidth can be measured for the forward path (from sender to receiver), the reverse path (from receiver to the sender), or both. *Capacity* is defined as the maximum possible amount of data per unit time a network can deliver along a path and the *narrow link* is defined as the hop with the minimum capacity

through that path. *Available bandwidth*, on the other hand, is the amount of data per unit time that is currently unused or available over the network path and the *tight link* is described as the hop with the minimum available bandwidth in the path.

The available bandwidth is a time-variant measurement and to be meaningful should be averaged over time and measured quickly in order to get an accurate and relevant estimate. Network end users can only estimate available bandwidth through the end system unless they have access to intermediate devices (such as routers) to gather network statistics. For the purpose of this thesis, it is assumed that an available bandwidth estimation tool provides end-to-end available bandwidth estimates to the proposed fuzzy logic controller. However, the type of tool and details are not specified.

2.2 Transport Layer Protocols

As described in RFC 768 [17], *User Datagram Protocol (UDP)* is a transport layer protocol used to send packets from one system's application port to another's application port. UDP differentiates from Transmission Control Protocol (TCP) in that it is an unreliable protocol that does not ensure packets reach their destination. It does not provide sequencing services that would ensure packets arrive in the order requested by an application nor any type of flow or congestion controls. It includes a header used to carry the source port, the destination port, the length, and the checksum to ensure there are no errors in the data. Some applications use UDP to save processing

time with a reduced header (compared to TCP) and less delay caused by ordering the packets at the receiver or retransmitting packets because of errors during transmission time.

Transmission Control Protocol (TCP) is another transport layer protocol that is defined in RFC 793 [24]. TCP is a widely used protocol that has gone through several versions to enhance its abilities. TCP is particularly suited for data applications as it guarantees the order and delivery of the data. It does not work efficiently for time-sensitive application information because includes controls which can restrict an application's throughput. TCP includes mechanisms for congestion and flow control. TCP includes an Additive Increase and Multiplicative Decrease (AIMD) algorithm which governs TCP to allow it to be friendly towards other flows and back off when congestion occurs. As shown in Figure 2, TCP goes through several phases including slow start and congestion avoidance. If there is no congestion detected, TCP increases a throughput parameter, the congestion window variable (cwnd), in an additive manner. If congestion is detected, it reduces the variable multiplicatively.

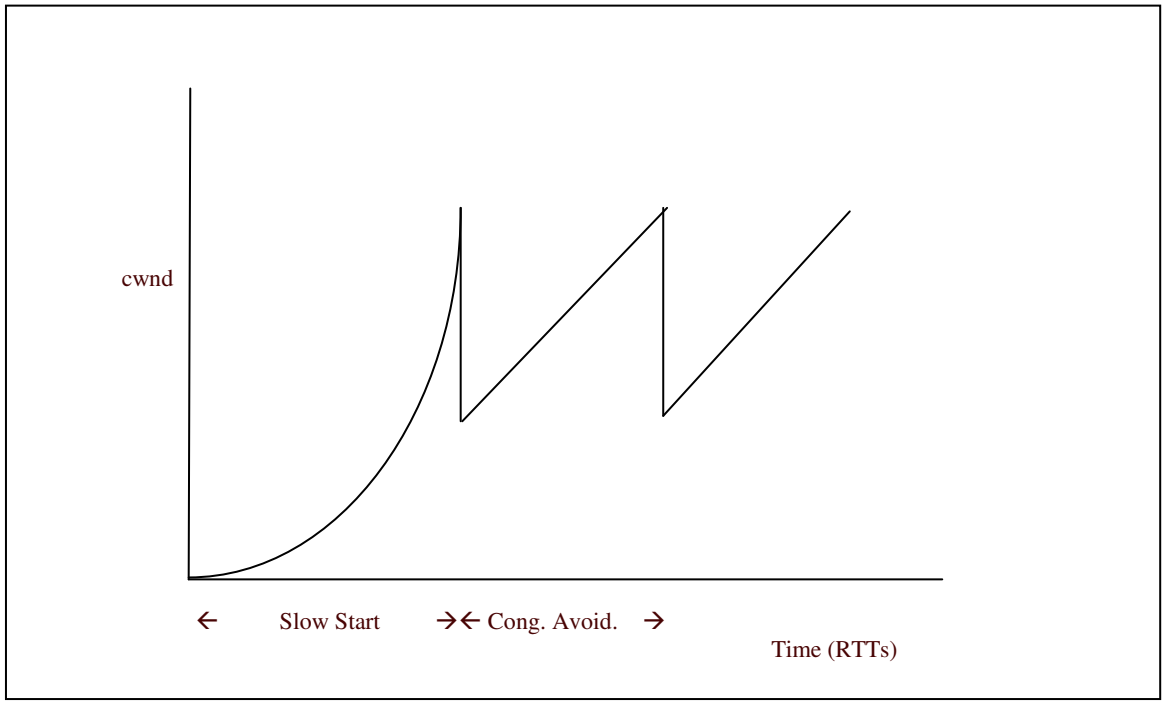

Figure 2 – TCP Congestion Control Phases.

2.3 Data and Streaming Media Applications

The applications that traditionally utilize UDP are *streaming media* applications. Streaming media is defined in Austerberry [2] as digitally encoded files delivered over networks that are delivered to the receiver's media player in real time. This means it is being received at the same rate as it is sent and there is no need for intermediate storage of the content. Applications include Internet audio players, videoconferencing, and IP telephony among others. TCP includes features, such as reliability and in-order delivery that aren't necessary for these applications. Video and voice applications prefer the timeliness of delivered packets to the delivery being guaranteed and in sequence. Streaming applications also prefer less variation in the rate of incoming packets. TCP will oscillate its delivery due

to network congestion more than UDP will because of the Additive Increase Multiplicative Decrease (AIMD) semantics.

Voice and video applications can experience service degradation when transporting information using network resources. They can suffer from delay, jitter, out-of order arrivals, and packet loss. *Jitter* is the variation in time between arriving packets. Jitter can cause noise or undesired effects in audio and perceivable visual effects in video. This is generally caused by network congestion. Delay is the time it takes for the data to leave the sender and arrive at the receiver. This time is a sum of the transmission delay (the time it takes to put the bits on the wire or take them off), processing delays (the time needed for bundling the data into packets, forwarding the packets to their proper destination), and queuing delays (time spent waiting to enter the link). Long delays can affect a video and voice application's perceived quality since these applications require real-time or near real-time transport. Data applications are elastic in how long delivery of the packets can take and they require all of the packets to be delivered. Video and audio streams are inelastic in the packet delivery time and with a robust encoding/decoding stream can handle some small packet loss.

2.4 Fuzzy Inference Systems

This thesis includes a Fuzzy Inference System (FIS) to aid in decision making. Fuzzy Inference Systems are based on Fuzzy Logic. Fuzzy Logic was introduced in Zadeh [28] as an alternative to traditional data processing and control without requiring crisp set membership. For example, a room with a traditional two valued on/off light switch may be

either light or dark. The values for a room brightness value would be either in the light set or not in it. Instead of requiring crisp set membership, fuzzy sets can include partial members. For example, a room with a dimmer light switch may have values like slightly dark or very bright. The values for the room brightness could then be partially in the light set and partially in the dark set. In Zadeh [28], it was noted that humans do not require crisp memberships and can handle complex reasoning. Also, not all real world input is crisp and clearly defined as in or out of a set.

As described in [25], Fuzzy Inference Systems (also known as Fuzzy Logic Systems) are mappings of input data into output data using fuzzy logic. Membership functions are the mapping of points within a range to a membership value (within a fuzzy set) from 0 to 1. Fuzzy logic operators perform similar functions as Boolean operators except that they can return multi-valued responses as opposed to a simple 0 or 1. Fuzzy Inference Systems utilize if-then-else rules with the inputs to resolve or defuzzify to an output. The fuzzy inference system then takes each input value, evaluates the rules in parallel using the fuzzy logic, and determines an output value.

Fuzzy logic maps well to the linguistic and subjective nature of expert knowledge and "rule of thumb" solutions to problems. For this thesis, the fuzzy nature of the inputs, the ability to map the linguistic terms of the problem to the system, and the number of input variables led to the decision to utilize a fuzzy inference system to determine the optimum send rate. The inputs for the problems in this thesis, such as available bandwidth, are considered fuzzy since they do not have clear, clean set members. For example, 1 Mbps available bandwidth may be considered "high" in one application context but not in another.

2.5 Current Problems in the Internet

In Floyd and Fall [5], *congestion collapse* is defined as occurring "when an increase in the network load results in a decrease in the useful work done by the network." As more packets are submitted into the network, there is less accomplished. Congestion in general is caused from a scarcity of bandwidth. Congestion collapse occurs from packets being sent into a network that doesn't have enough bandwidth to handle them. These packets add to the congestion, consume network resources and never reach their destination. As the traffic increases, there also exists the possibility of extreme unfairness against TCP flows.

In [5], the problem is divided into five categories of congestion collapse: classical congestion collapse, congestion collapse from undelivered packets, fragmentation-based congestion collapse, congestion collapse from increased control traffic, and congestion collapse from stale or unwanted packets. This thesis is concerned with congestion collapse from undelivered packets.

Classical congestion collapse occurs when packets are retransmitted when they don't need to be. The cause of this type of congestion collapse can and has occurred from TCP retransmitting packets even though they are arriving intact at the destination or still are in flight to the destination because of incorrect timers or missing or invalid congestion control. The effect of this is to provide a steady state of the network being inefficiently used for unnecessary TCP retransmissions. This problem is solved with improvements in TCP timing and congestion control.

Congestion collapse from undelivered packets occurs when a node inserts packets into the network that will not be able to deliver the packets but loses them at some link along the path. The insertion of these packets into the network accomplishes no work but does add to the congestion of the network. The network will be congested only as long as the undeliverable packets continue to be inserted. The packets consume resources without accomplishing work. UDP packets can produce this type of congestion by not including congestion control (as TCP does) and by inserting packets into the network that have no guarantees of being delivered. This is the primary type of congestion collapse of interest for this thesis.

Fragmentation-based congestion collapse occurs when there is an inconsistency in the size of the data at the link layer and the higher network layers. When some of the link layer frames (which make up the packet) are received but others are lost, it causes resources to be spent on retransmitting data to be able to assemble the packet at the receiver correctly and completely. There are mechanisms, such as Path MTU (Maximum Transmission Units) that reduce the packet fragmentation that causes this type of collapse by determining the largest transmission unit possible along the entire path and setting the size accordingly in the packet before inserting the packet into the network.

Congestion collapse from increased control traffic occurs from an increase in control traffic (such as routing table updates) which can occur from an increase in network load and congestion. As the network becomes even more congested, more control traffic packets may be inserted. This cycle can cause a decrease in the amount of useful work completed from the available bandwidth. As described in Kelly *et al.* [10], as the load on the network

increases past the available bandwidth (for example with additional packets from unresponsive flows or control packets), the amount of useful work done can suffer. Figure 3, taken from [10] shows the aggregate throughput (which at a maximum for the timeframe could be 300000 packets) versus the offered load. As the offered load increases past the capacity of the network and K links, the aggregate throughput suffers.

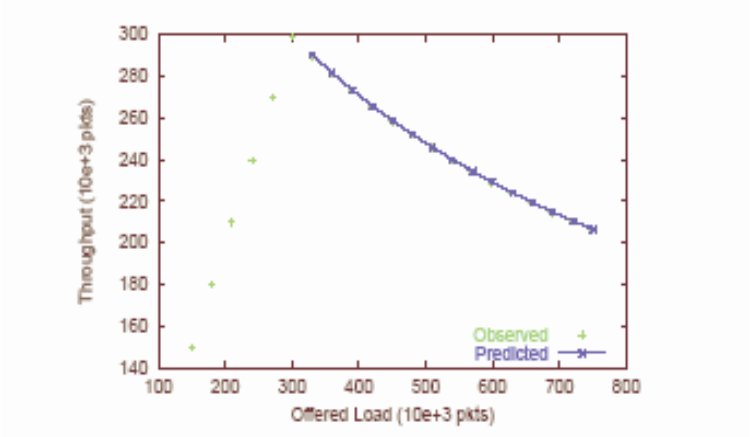

Figure 3 – Offered Load vs. Goodput for K = 2 (Kelly et al. [10])

The final type of congestion collapse is congestion collapse from stale or unwanted packets. Stale packets can cause congestion collapse by using network resources when the users are no longer interested in the data. This can occur in the case of unlimited network buffers, where the packets have such long queue delays that by the time they reach their destinations, the user no longer requires the data. The network can also be congested with packets that are "pushed" to a destination node even though they were not requested. Both of these scenarios inject packets into the network that consume resources without providing useful work.

Another important issue in the Internet is the *TCP-friendliness* problem. Discussed in [5], the problem can arise from applications using a transport layer protocol without congestion control, such as UDP. It occurs when UDP and TCP share a congested link where the bandwidth is not large enough to handle both the UDP and the TCP flows. It is well known that in these circumstances UDP grabs the channel capacity at the expense of the TCP flow. While UDP will continue taking as much bandwidth as it needs, TCP will reduce its flow after detecting congestion. TCP then receives an unfair amount of the bandwidth and UDP is considered un-friendly to TCP.

2.6 Previous Work

There are many possible solutions for the congestion collapse problem and congestion control in general. Congestion collapse solutions consist of several high-level components. First, how the solution determines there is congestion: does it use receiver-side feedback, available bandwidth estimation etc. Second, how the solution adjusts the flow of data: does it use windows-based flow control or rate-based flow control. This can further be defined by how the solution adjust the flow based on the AIMD rule or use a TCP model. TCP model solutions attempt to model the response on a model of TCP traffic and act as TCP would without the reliability mechanism. Finally, where the solution is implemented. The location of the solution can be in the router or intermediary network systems; at the sender or at the receiver system. The location can be further defined by OSI (Open Systems Interconnection) layer location. For example, a solution can be implemented at the Application layer or the Transport layer. Figure 4 below, shows the search space

categorization of the solutions broken down by flow control, location of solution, and incentives. The solutions can also be broken down by the amount of time they take to react. The bandwidth restriction-based solutions, per-flow scheduling and end system-based solutions can react relatively fast (within seconds, milliseconds or even microseconds) while pricing incentives may take hours or longer to take effect.

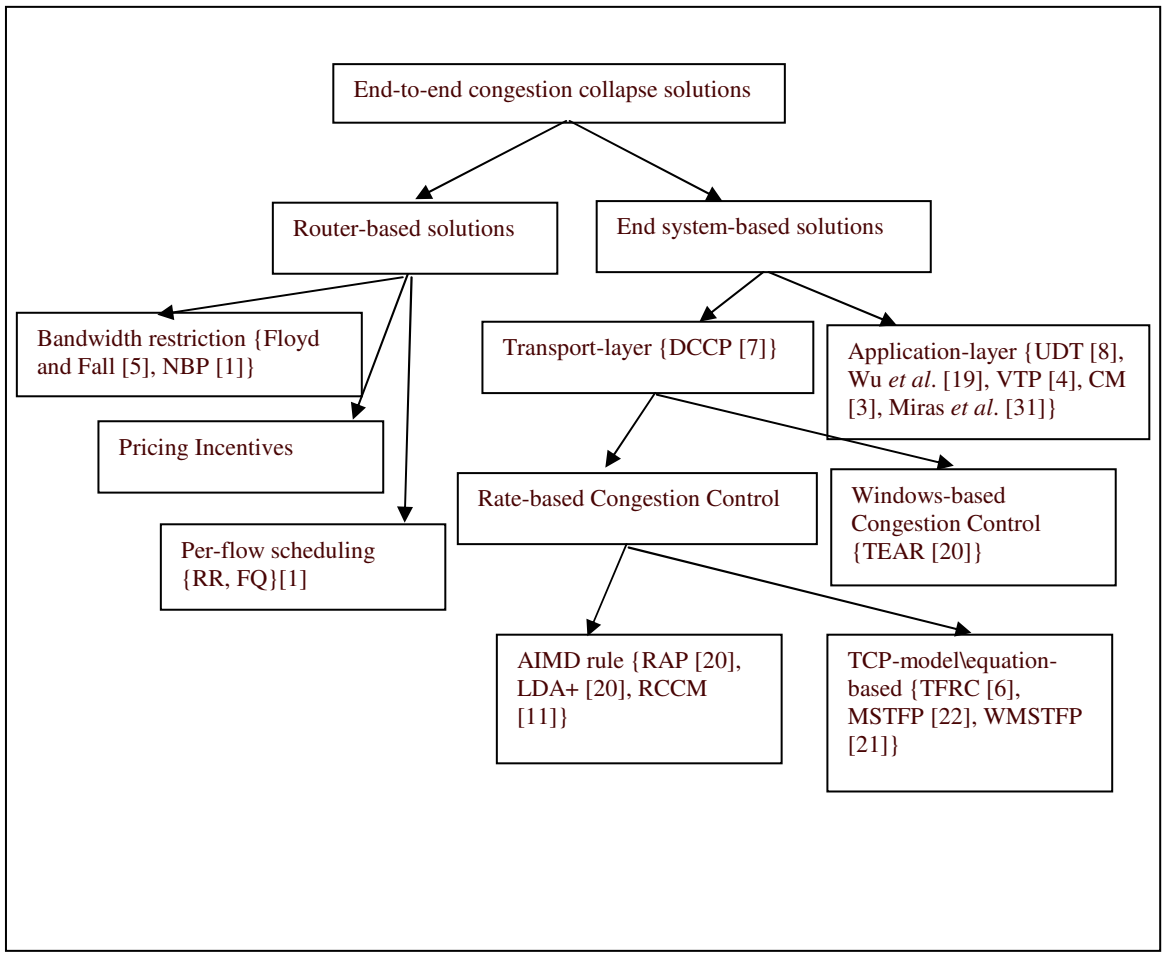

Figure 4 – Solution Space.

2.6.1 Router-Based Solutions

In Floyd and Fall [5], there authors consider a router based solution. For this to work, the network nodes must all react to congestion by detecting congestion and adjusting the bandwidth used to meet the available levels. Congestion collapse prevention at the routers has been shown to not completely prevent congestion collapse. With a number of flows that do not provide congestion control and do not limit what they send during congestion, congestion collapse can still occur. In the paper, the authors show that the underlying factor producing the congestion collapse from undelivered packets is UDP flows without end-to-end congestion. The routers can provide some assistance by detecting congestion and sending signals that congestion is occurring. The routers can also control greedy users that choose to not use congestion control. The routers can also isolate the flows that provide end-to-end congestion control. The router side solution space for the congestion collapse problem includes:

1. Per-flow scheduling – relies on social incentives. This signals the application to govern its network usage in an effort to be fair.
2. Router mechanisms – Service incentive of restricting the bandwidth of disproportionate flows.
3. Financial and pricing incentives – the problem is how to react fast enough to meet rate of growth of unresponsive flows.

The author's argue that it isn't practical to rely on end users to use end-to-end congestion control and that social and financial incentives are difficult to realize, but that

control must be managed by the infrastructure itself. This requires compliancy from all of the routers. There are two types of router controls suggested: flow identification and regulation or per-flow scheduling. Per-flow scheduling includes Round Robin (RR) and Fair Queue (FQ) scheduling. The authors argue that the additional implementation complexity (over the simpler First Come First Serve (FCFS) scheduler), flow aggregation issues (where one source can create many flows to a single receiver and in effect receive more bandwidth), and unresponsive flow congestion collapse possibilities make per-flow scheduling a less attractive option.

The paper suggests a router-based bandwidth restriction solution. The authors propose to use router mechanisms that identify and regulate certain misbehaving flows as a solution. It is suggested that this regulation would provide the incentives needed to remove the congestion collapse possibility. In the paper, they describe how to identify the flows but not exactly how to regulate them once identified. The mechanism would look for three types of flows: unresponsive, non-TCP friendly and flows that used disproportionate levels of bandwidth. Unresponsive flows are defined as flows that do not have a decreasing arrival rate at the router when there is an increasing packet drop rate. Routers would identify non-TCP-friendly flows using an equation from the paper that would produce a table of steady-state packet drop rates mapped to maximum flow arrival rate. A TCP-friendly flow is one that would act the same to other TCP flows as a TCP flow would. If a flow is using more than the coordinate arrival rate, the router would be allowed to restrict that flow's bandwidth. There is not a guarantee that TCP-friendly flows will be friendly between each other. For example, a flow with a large RTT may perform worse than a flow with a short RTT since

TCP increases its window using the RTT. Flows using disproportionate levels of bandwidth would be identified by equations for flows using a disproportionate share of the bandwidth and a high arrival rate relative to the level of congestion. One issue with this system, as noted in Albuquerque *et al.* [1] is that current methods to identify unresponsive flows do not always work. Also, for the system to work, it requires changes to all the intermediate systems to govern the flows correctly.

An example of another router-based bandwidth restriction solution is Network Border Patrol (NBP). In [1], the paper introduces NBP as a congestion avoidance mechanism router-based solution. The mechanism is implemented at the edge routers of a network to restrict packets being sent into the network that are likely dropping packets before they reach the destination. The authors argue against per-flow packet solutions claiming they are complicated (compared to First In First Out), expensive and not currently global solutions which allows for congestion collapse to still occur. Instead they argue for routers to communicate amongst each other to identify flows which may be inserting undeliverable packets into the network. The authors use the edge routers in attempt to push the complexity to the edge of the network. The solution identifies flows using ingress and egress routers respective to a flows entry and exit points within the network. The mechanism identifies flows inserting undelivered packets by comparing the rate that packets enter the network against the rate they leave it. The tool assumes congestion is occurring if the exit rate is a certain degree less than the entry rate. After feedback has been received and congested flows identified, the rate control algorithm reduces the rate of the flow. As noted in Floyd and Fall

[5], router-based solutions can aide in reducing congestion control but not eliminate the possibility.

2.6.2 End System-Based Solutions

There are currently several ways to detect congestion from an end system. One way is using sender side windows-based flow control and adjusting the window size based on receiving positive or negative ACKs (acknowledgements) from the receiver. TCP is a sender-based rate control and window-based congestion control that estimates RTTs (Round Trip Times) to determine if congestion exists. Packet loss is indicated by a timeout. Congestion can also be inferred from the available bandwidth. If available bandwidth is trending to decrease, congestion is increasing. If there is no available bandwidth, congestion exists. Solutions can probe the network for available bandwidth or they can also track history and detect trends to predict congestion. Congestion can be inferred from packet loss, whether explicit (e.g. with Explicit Congestion Notification ECN) or implied. UDP is a connectionless protocol and thus packet loss cannot be used with UDP alone since the sender has no guarantee of receiving feedback from the receiver about packet loss.

Next we investigate application layer solutions. In Gu and Grossman [8], the paper describes a new protocol UDP-based Data Transfer protocol (UDT) as an application level protocol solution to congestion control. The protocol is an addition above UDP that provides reliability and congestion control. It accomplishes this by combining both rate and windows-based congestion control mechanisms. The protocol uses the rate control to determine the performance and the windows-based control is used to ensure TCP-friendliness. The

bandwidth estimation used to determine the rate increase is a capacity measurement done using pairs of probing packets and a weighted average. The solution does not account for providing a smooth rate for the applications.

In Balk *et al.* [4], the authors introduce Video Transport Protocol (VTP). This is a rate-based application layer solution that includes receiver-side bandwidth estimation using a tool called NetPeer. The protocol requires there to be special code running at both the sender and receiver end to enable the protocol. The paper specifically restricts the protocol to MPEG-4 video. The bandwidth estimation is done using acknowledgements (ACKs) and RTTs and averaging the samples taken. The solution will also verify increases in bandwidth before adjusting the rate to ensure that extra bandwidth is actually available. The rate adjustment is different than TCP in that it will not reduce its rate as the AIMD algorithm specifies but will reduce it to a level that with the bandwidth estimation it believes the network can handle. The protocol also includes schemes to modify the video encodings to best match the available bandwidth and produce the best quality. The protocol is a media specific solution.

A multi-layer solution is described in Wu *et al*. [19] which discusses how MPEG-4 can be transported over the Internet. This scheme uses a receiver side packet loss ratio deduced from the sequence numbers to determine available network bandwidth. It utilizes AIMD (Additive-Increase Multiplicative Decrease) to adjust the rate in response to congestion feedback. It utilizes RTP (Real Time Protocol)\RTCP (Real Time Control Protocol) messages to communicate congestion feedback. The solution does not utilize the network in any portion better than TCP does.

In Balakrishnan *et al*. [3], the authors introduce an end systems framework to solve the congestion collapse problem. Their solution, named the Congestion Manager (CM), is a generic architecture that sits between the application and below the transport layer to provide an Application Program Interface (API) that allows the applications to learn about the network condition and also notify other layers about the perceived network condition. The solution includes windows-based congestion control, receiver-side feedback for bandwidth estimation, and an AIMD rate controller. The solution estimates the bandwidth using probes that are sent to gather information from ACKs and Explicit Congestion Notification. One advantage of this solution is that all of an end system's connections can share information about the network and share the bandwidth more fairly between themselves. This information can be shared across the flows and protocols (e.g. TCP and UDP flows). The centralization also encapsulates the congestion control and frees the application above from this responsibility. The solution requires specific receiver-side code to accommodate the windows functions of non-TCP flows. A disadvantage noted by Kohler *et al*. [13] is that the CM is limited to a single congestion control mechanism (as opposed to DCCP's flexible approach) and that there can be middlebox traversal issues.

In Miras *et al.* [31], the authors introduce a method to smooth the source rate of a live internet video stream and increase the user's perceived quality. The author's utilize input from a TCP-Friendly congestion control mechanism, artificial neural networks for predicting video quality, and a fuzzy controller to consider both the send rate for TCP Friendliness and the quality of the video. While this method does include congestion control, it is video specific and relies on buffers to improve the perceived quality.

The congestion collapse problem has also been investigated at the transport layer. In Kohler *et al*. [13], the Datagram Congestion Control Protocol (DCCP) is specified and in Kohler *et al*. [12], the design of the protocol is described. The protocol is intended to provide congestion-control for unreliable flows of applications that put more emphasis on timely delivery of the data rather than orderly and receiver acknowledged data. Example applications given are streaming video, Internet telephony, and on-line games. These applications have a receiver that wants the data arriving to be the latest data and doesn't need the sender to resend any lost packets, since by the time they are resent they wouldn't be useful any longer.

The DCCP protocol attempts to provide the services required by the delay-sensitive applications while not subjecting the Internet to the congestion collapse possibility as the UDP protocol (currently used for these applications) does. The design of the protocol intended to make it as simple and general as possible using only the minimum overhead and include only the minimum of required functions. The design is also intended to be flexible and include a large number of bits for options to allow the protocol to be used with future technologies.

Here is a brief synopsis of the DCCP functionality and characteristics described in the paper. The protocol is *unreliable* in that the packets are not resent if they do not reach the receiver. Although the packets are not sent, *acknowledgements* are included so that the sender can be informed by the receiver of congestion and react appropriately. The protocol allows the sender to distinguish whether the packet reached the receiver's application or was dropped in the receiver's buffer. Similar to TCP, the protocol utilizes a *connection*

handshake for setting up and tearing down connections. This allows the protocol to get through middleboxes such as firewalls and Network Address Translator (NAT) boxes similar to how TCP does and better than UDP. Traversal is made more probable by using a single connection as opposed to multiple randomly generated ports. The *choice of congestion control mechanisms* is an option and can be different for each of the half-connections (sender-to-receiver and receiver-to-sender) supported. There is *Explicit Congestion Notification (ECN)* support. *Path MTU Discovery* is also included to eliminate the potential for fragmentation-based congestion collapse. The protocol also includes *options*, for flexibility, and *reliable feature negotiation* so that the sender and receiver can agree on the best supported methods for the connection. The DCCP protocol does not directly and completely solve the congestion collapse problem without specifying TCP-Friendly Rate Control (TFRC) which is discussed later.

In Widmer *et al*. [20], the paper surveys TCP-friendly congestion control solutions. The survey includes AIMD and windows-based congestion control schemes. The Rate Adaptation Protocol (RAP) is a rate-based protocol that uses acknowledgements (ACKs) to detect packet loss and infer the RTT. It adapts to the network status with AIMD and increases in times without congestion by 1 packet per RTT. The paper suggests that RAP may act more aggressive than TCP since it does not take into account timeouts. The Loss-Delay Based Adaptation Algorithm (LDA+) utilizes feedback messages from RTCP and controls the rate using something similar to AIMD. For, example, the algorithm will increase the rate of a low bandwidth flow faster than a high bandwidth flow. The paper notes that relying on RTCP, which sends feedback to the sender infrequently, may be slow to react.

TCP Emulation at Receivers (TEAR) uses a congestion window at the receiver and attempts to determine how TCP would adjust the rate. The paper notes that the protocol can only roughly estimate timeouts but does act friendly to TCP without the saw-like behavior of TCP. One issue with schemes that try to be TCP-friendly is that they must also degrade its throughput with high RTTs since TCP does this.

Another TCP-like congestion control and AIMD rule solution is described in Kim *et al*. [11]. The authors introduce a complete transmission scheme including a TCP-friendly end-to-end congestion control mechanism and available bandwidth estimator known as Receiver-based Congestion Control mechanism (RCCM), an encoding and smoothing component at the sender, and quality recovery tools at the receiver. This solution is a video specific solution based on the ITU-T H.263+ scheme. It consists of trying to reduce sending of packets that will not reach the receiver and trying to adjust the quality or resolution of the video based on the available bandwidth the sender has. Although this work is not covering all of the aspects of the video streaming problem, it is interested in the RCCM mechanism.

The RCCM mechanism is a mechanism that produces feedback to the sender in the form of a bit budget or available bandwidth the sender can use. It is a video specific tool created with video rate and error recovery issues. In order to provide a smoother transition in sending rate, RCCM relaxes the AIMD rule and utilizes equations adjusting the rate with steps like weighted temporal smoothing. RCCM estimates the available bandwidth using inter-arrival time, size and loss of each packet or using a congestion degree measurement. While this solution does have advantages, it is media specific.

Finally we discuss the TCP model and equation-based schemes. In Kelly *et al*. [10], the authors attempt to create models to investigate how traffic patterns, topologies, transport protocols, and other variables affect high consistent levels of packet drops. The authors introduce dead packets (a packet that never reaches its intended destination), duplicate packets, dummy packets (packets that carry nothing requested or useful), and fragments of packets. A second goal of the paper is to investigate how the variables mentioned above affect the number of dead packets in the network. The paper introduces a number of equation-based theoretical models for a number of scenarios. The difference between the model predications and simulated results are then analyzed to validate the correctness of the models.

The results of the paper showed that generally high consistent packet loss rates can occur with greedy senders sharing either a FIFO or FQ link. The paper admittedly could not find an accurate general model for system equilibrium with general topologies and greedy sources although it did find specific validated models such as goodput prediction in a Random Early Detection (RED) queue management in a cyclic topology. It further specifically found that in a cyclic network, as the load on the network is increased, the sum total number of packets reaching their destination and doing useful work can decrease. This decreased dead packet ratio increases (closing to 1) as more links are added. While the paper did create and validate many models for specific network scenarios, it does demonstrate the need for congestion control to remove the congestion collapse possibility for all sending sources regardless of the network scenario (topology, queue management, links, bandwidth, etc.).

In Floyd *et al*. [6], the authors investigate a model based algorithm for end-to-end congestion control. Their solution introduces TFRC (TCP Friendly Rate Control) to provide congestion control for real-time applications in order to provide an incentive for applications to use it while reacting more beneficial to other flows in the network. The algorithm tracks a loss event rate as opposed to reacting to a single lost packet. It utilizes an equation to mimic closely what TCP would do in the same situation to achieve TCP-friendliness. It attempts to match the throughput that a TCP flow would have in the same situation. The solution attempts to meet the requirements of real-time applications while remaining friendly to other network needs. Real-time applications react poorly to drastic changes in bandwidth by making perceivable effects to the user such as video becoming static or pixilated. The paper attempts to provide a smoothing of the bandwidth to reduce this effect. TFRC is giving an upper bound to the rate sent out into the network.

Equation-based congestion control attempts to form an equation mimicking the TCP response function. It attempts to match how TCP would react in a certain network environment with certain network parameters. The solution does not grab all available bandwidth greedily but rather responds to congestion while attempting to keep a relatively smooth rate for the application. The solution measures the loss event rate over a long period of time to provide a less fluctuating available bandwidth measurement. The loss event rate is used as opposed to a packet loss rate. It is used to predict future packet losses. The loss event is packets lost within a round-trip time and the loss interval it is measured over is the number of packets between loss events. The loss event rate is then described in Handley *et al*. [9] as the number of loss events as a fraction of the number of packets transmitted.

In the papers, Zhang et al. [22] and Zhu *et al*. [23], the Multimedia Streaming TCP-friendly Protocol (MSTFP) is introduced. This protocol is further extended in Yang *et al*. [21] to adapt to the wireless Internet with Wireless Multimedia Streaming TCP-friendly Protocol (WMSTFP). A major addition is WMSTFP adds the ability to differentiate between errors in the wireless hop and congestion which would be adapted to. The solution sits at the application and transport layers and performs packet loss ratio, RTT, and available bandwidth estimation and sender rate adjustment. The protocols use the receiver's feedback to get packet loss rate, RTTs and timeouts to estimate the available bandwidth. With this estimation the sender decides what rate to send using a TCP model to ensure TCP-friendliness. It integrates the historical data to provide a smooth rate for the applications. The solution has some components that are specific to MPEG4 video. It adapts the encoding during good network conditions to provide more bits for better resolution and reduces the amount of bits in bad network conditions. This allows the user to still get some video in bad network conditions when without modifying the budgeted bits may have provided more perceivable video annoyances.

With TFRC, MSTFP and other model based solutions attempting to match what TCP would do in a situation, they will be limited to the same functionality as TCP. The issue with the approach utilized in strict TCP-friendly solutions is that they focus primarily on TCP-friendliness and may be missing out on available bandwidth and effectively waste resources. TFRC will dictate an upper bound rate which should not be overrun in order to remain strictly TCP-friendly but this does not mean that it utilizes the entire available bandwidth and

thus may sacrifice utilization of the network to meet friendliness constraints. They may not utilize the bandwidth as well as other solutions but will remain friendly to other TCP flows.

In addition to the other solutions, there have also been recent attempts to use fuzzy logic to perform congestion mitigation and congestion control as described in Ghosh *et al.* [29] and Douligeris *et al.* [30]. In [29], the authors survey many application of fuzzy logic to telecommunications and include solutions for congestion mitigation and congestion control. In [30], the authors describe a particular system to shut down flows that violate agreed to rates. These solutions are specific to ATM networks and not generally applicable to the best effort environment of the internet.

As described above in this chapter, many solutions have been attempted for the congestion control and TCP-friendliness problems. Both router-based and end-to-end solutions have been proposed to solve the problems. However, no fuzzy logic based controller has been proposed so far to address these problems at the end system for a best effort environment. A controller utilizing a Fuzzy Inference System could be used to find an optimum output (a send rate) given fuzzy inputs (such as network conditions and the TCP response). This is the approach taken in this thesis.

Chapter Three

Solution

3.1 Solution Requirements

In order to provide a solution for the congestion collapse, TCP-friendliness, and smoothness problems that is practical and utilized by applications, specifically applications that currently choose protocols without congestion control, there must be an incentive for the applications. For the applications or end systems in general to utilize a congestion control mechanism, they must find that the application results (e.g. video or voice) are perceivably acceptable. The applications must be provided adequate bandwidth with smooth bandwidth transitions that allow for minimal perceived effects. As modeled in Figure 5, the proposed solution must then be an optimization of several factors: the application requested rate, a network friendly mechanism that doesn't steal bandwidth from competing flows, a minimum variation in the adjustment of the sending rate, and network conditions such as current available bandwidth, Packet Loss Rates and Round Trip Times.

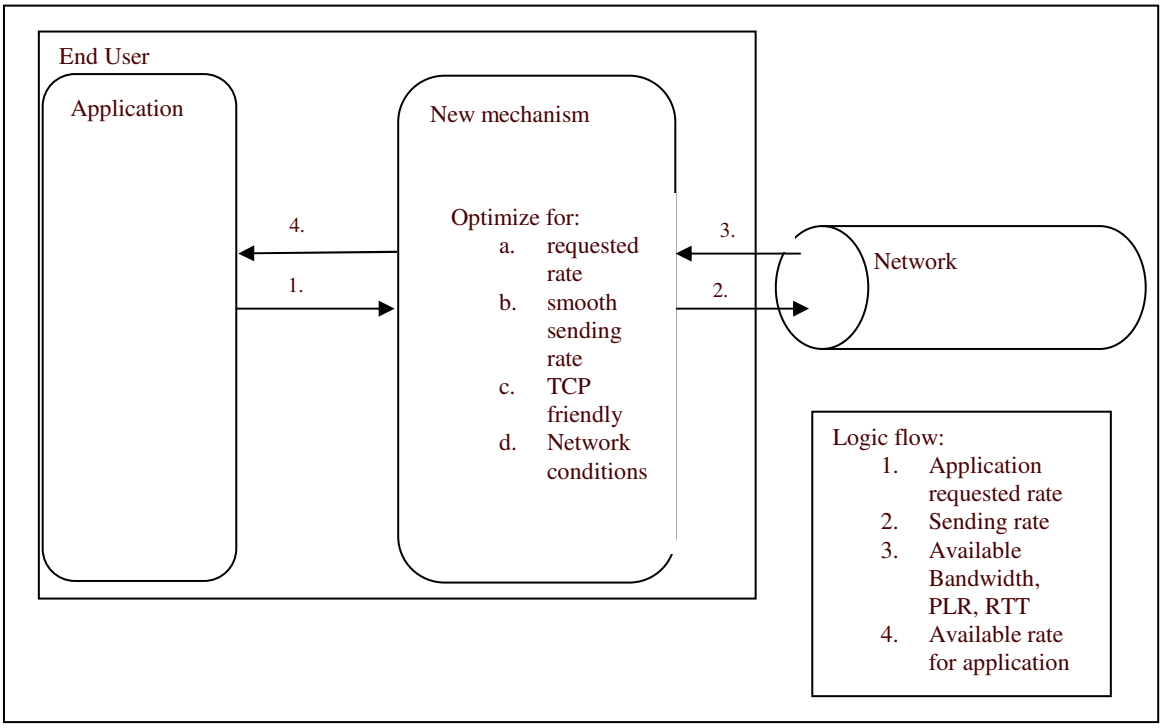

Figure 5 – Solution Model.

The previous attempted solutions meet only a partial set of the requirements for bandwidth, congestion control, friendliness and smoothness. Some solutions met the applications requested rate but would do it at the expense of being unfair to other flows. Other solutions may meet some requirements but were at the application level and would not work for all types of flows. Other flows met the friendliness requirements but would underutilize the network. While the latest solutions are friendly to other flows and smooth the fluctuations for better perceived quality, they do not utilize the network resources fully.

3.2 Solution Design

This thesis proposes to smoothly meet the requested rate for all types of flows while remaining friendly to other flows and utilizing the network resources efficiently. While it is

not possible to meet the requested rate in all scenarios the solution should determine the optimal rate considering other flows, network conditions, smoothness of the provided rate and providing as much of the requested rate as possible. The solution will need to operate below the application level so that it can apply towards all IP applications. In order to understand the network conditions, the solution will need to gather measurements including the available bandwidth, Packet Loss Rates and Round Trip Times. However, the solution assumes that modules providing the information exist and do not dictate how or specify details about how these modules measure and report available bandwidth, Packet Loss Rate and Round Trip Time estimates. Rather, this thesis is concerned with the design of the controller, which based on fuzzy logic and all this data, will provide as output the appropriate sending rate that will satisfy the objectives set forth in Section 1.1.

As shown in Figure 6, the components of the solution include: the application requesting the bandwidth, an available bandwidth estimation tool, Packet Loss Rate (PLR) and Round Trip Time (RTT) estimation tools, the FIS controller, and the FIS itself. The PLR and RTT estimations are used to determine the TCP Response. Although other optimization tools may have also worked, fuzzy logic was chosen due to the fuzzy nature of the inputs, the ability to map the linguistic terms of the problem to the system and the rather large number of variables involved in the optimization.

The application, upon beginning or continuing a network flow, calls the FIS controller with a specified request rate. The FIS controller then gathers the necessary inputs including the available bandwidth, the parameters used in determining TCP's response and the history of the send rate issued. The controller then passes that information as input into

the FIS and receives the send rate in response. The FIS then returns that send rate to the application so that the application can then make the necessary adjustments (e.g. with UDP a reduction or increase in the rate of packet insertions into the network).

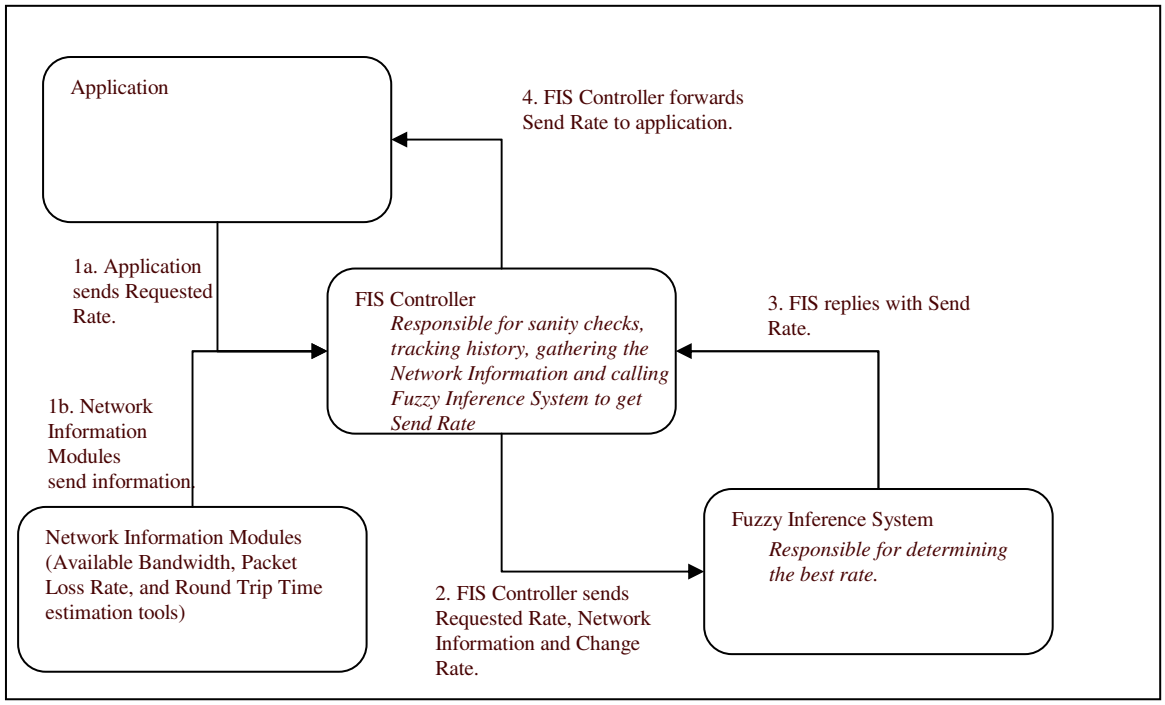

Fig. 6 – Solution Components.

3.3 Fuzzy Inference System Controller

The FIS controller is a program written in C to act as an intermediary function between the application and the FIS, collect the necessary input for the FIS, normalize the inputs and outputs, call the FIS and relay the response back to the application. The FIS controller requires some preprocessing of the input values. The controller derives the change rate input by subtracting the two most recent send rates given by the time elapsed between them. The change rate is then normalized as a value between -1 and 1 for the fuzzy inference

system where -1 indicates a large decreasing trend and 1 indicates a large increasing trend in the send rate. While the values from this equation can be larger than 1 and smaller than -1, the change rate input is truncated into the -1 to 1 range as containing the values within a given range increases the stability of the controller and the values would still be represented as quickly decreasing or quickly increasing. The function is shown below in Equation 2.1 where s_1 is the last send rate and s_2 is the second to last send rate and Δt is the change in time:

$$\frac{s_1 - s_2}{\Delta t} \tag{2.1}$$

The controller then determines the average sending rate response TCP would have using the TCP Response function from a formulation of Floyd and Fall [5] with the following function, shown in Equation 2.2, where *p* is the Packet Loss Rate, B is the packet size and R is the Round Trip Time:

$$\frac{\sqrt{3/2} * (B)}{R * \sqrt{p}} \tag{2.2}$$

The TCP Response and available bandwidth are both then normalized as a value between 0 and 1 using the requested rate. If the TCP Response is greater than the requested rate, it is truncated to 1 to allow the variable to fit into the FIS input range and still show that the TCP response is high. If the available bandwidth is larger then the requested rate, it is also truncated to 1 to stay within the input variable range and still represent a high available bandwidth. Once the inputs are processed and ready, the controller then calls the fuzzy inference system with the available bandwidth, TCP response, and change rate to receive the

send rate. The send rate is then saved as the last rate and multiplied as a percentage of the available bandwidth.

The FIS controller was written in C and includes calls to MATLAB C files which run the FIS engine. In order to run the FIS and FIS controller outside of MATLAB, the FIS was developed within the MATLAB environment and saved as a FIS file. The FIS file is a type of file that describes in detail the configuration of the FIS (e.g. the type of FIS, the input variables, etc.). The MATLAB stand-alone fuzzy inference engine, which is comprised of two files: fismain.c and fis.c, was compiled along with the newly created FIS controller. After the system has been compiled and run, the FIS controller can call the stand-alone engine with the FIS inputs and the FIS file as parameters to retrieve the output.

The FIS controller and the MATLAB files all had to be modified to enable the code to be run by a kernel. For example, the original MATLAB code utilized files to store parameters and data. Since many kernels do not allow or support file manipulation calls at the kernel level, this code had to be modified to store the data in variables or pass the data as parameters. While the entire C

code system includes many important functions, the following section of code provided focuses on the main pre-processing and post-processing function of the C language FIS controller:

```
/*********************************************************************
 Function to calculate sendRate
 *********************************************************************/

double getSendRate(double requestedRate, double availableBW, double RTT, double
packetDropRate, int packetSize)
{
```

```
double sendRate, changeRate, tcpResponse;
float elapsedTimeLong;
struct timeval currentTime, elapsedTime;

/* set variables */
_requestedRate  = requestedRate;
_availableBW    = availableBW;
_RTT            = RTT;
_packetDropRate = packetDropRate;


/* calculate change Rate */
gettimeofday (&currentTime, NULL);
if(&_lastSendTime == NULL) {
   changeRate = 0.0;
} else {
   timeval_subtract(&elapsedTime,&currentTime,&_lastSendTime);

   /* changed the struct into a long */
   elapsedTimeLong = (float)elapsedTime.tv_sec +
   ((float)elapsedTime.tv_usec/1000000.0);

   changeRate = (_lastSendRate - _lastSendRate2)/elapsedTimeLong;
}

/* normalize the changeRate */
if(changeRate > 1)  changeRate = 1;
if(changeRate < -1) changeRate = -1;

/* calculate tcpResponse - this is taken from Floyd's TFRC paper */
tcpResponse = (sqrt(3/2)*packetSize)/(RTT*sqrt(packetDropRate));

/* tcpResponse is in Bytes but we need bits */
tcpResponse = tcpResponse*8;

/* normalize inputs for tcpResponse and availableBW */
if((availableBW/requestedRate) <= 1) {
   _availableBW = (availableBW/requestedRate);
} else {
   _availableBW = 1;
}

if((tcpResponse/_requestedRate) <= 1) {
   tcpResponse = (tcpResponse/requestedRate);
```

```
    } else {
       tcpResponse = 1;
    }

    /* call fis */
    sendRate = fismainEmulation("fis_input", "CongestionControl.fis",
    tcpResponse, changeRate, _availableBW);

    /* update lastSendRate and lastSendTime for change Rate calculation */
    _lastSendRate2 = _lastSendRate;
    _lastSendRate  = sendRate;
    _lastSendTime  = currentTime;

    /* un-normalize sendRate output */
    sendRate = sendRate * availableBW;

    /* no need to go past requestedRate */
    if(sendRate > _requestedRate) {
       sendRate = _requestedRate;
    }


    /* return sendRate */
    return sendRate;

}
/* end of getSendRate() */
```

3.4 Fuzzy Inference System Details

As shown in the previous section, the FIS system consists of three input variables, one output variable, and the Fuzzy Inference System engine (which is a Mamdani type with a centroid defuzzification). The input variable membership functions are all curved functions while the output variable membership functions are linear. The variables include the following:

Inputs:

1. availableBW – Available Bandwidth which is normalized as a percentage of the requested rate ranging from 0 to 1.
2. tcpResponse – is normalized as a percentage of the requested rate ranging from 0 to 1.
3. changeRate – is also normalized as a percentage of the requested rate but can range from -1 to 1.

Outputs:

1. sendRate – is a value ranging from 0 to 1. Note that this value is multiplied times the availableBW before being given to the application.

The membership function graphs display the ranges of values that can be accepted for inputs and delivered as an output. Figure 7 shows the input variable for available bandwidth. The values for this input range from 0 to 1 where 0 represents a low value or no available bandwidth and 1 represents a high value or the maximum available bandwidth. These membership functions were chosen as horizontally and vertically symmetrical curves splitting the mapping values in an effort to elevate the middle sections (slightly more than a straight line would do) and add weight to the variable.

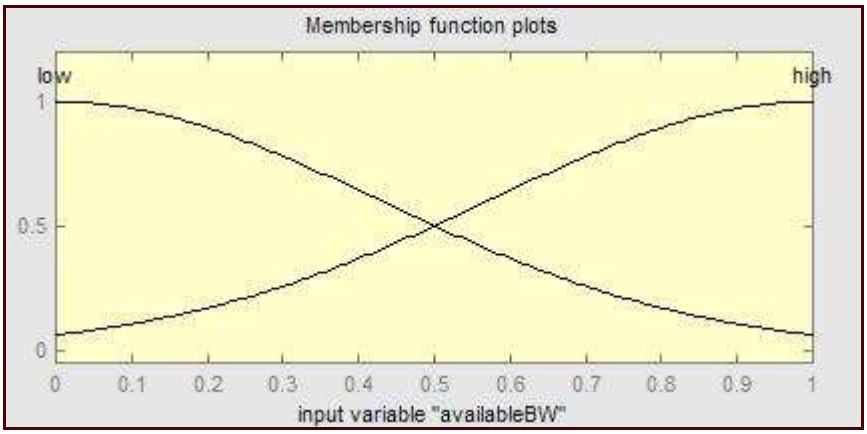

Figure 7 – The Available Bandwidth Membership Functions.

The next input variable, shown below in Figure 8, is for the TCP response input. The values for this input variable range from 0 to 1 also where 0 represents a low value or TCP flow not utilizing any of the path bandwidth and 1 representing it using the maximum or a high value. These membership functions were also chosen as curves for the TCP response input with the emphasis given to the low and high ends over the middle section to allow the other input variables to play more of role in determining the send rate when the TCP response was in the middle of the range.

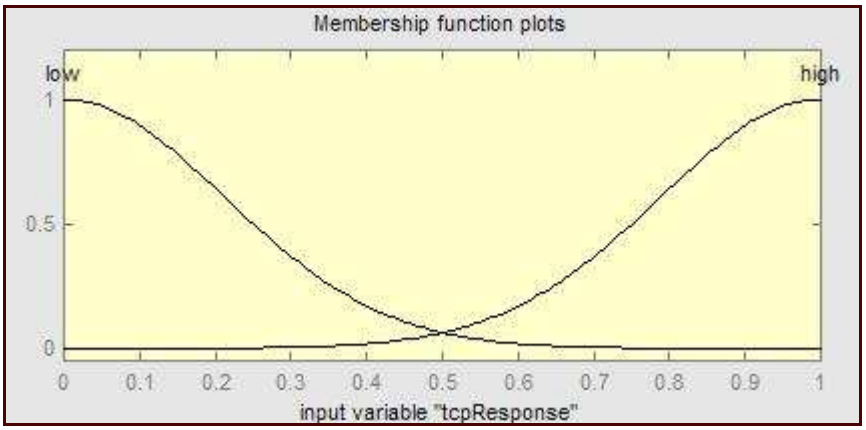

Figure 8 – The TCP Response Membership Functions.

The final input variable is for the change rate. As shown in Figure 9, the range is from -1 to 1 where -1 represents a decreasing change rate (the send rate has been governed more and more) and 1 represents a increasing trend (the send rate has been less and less restricted). These membership functions were also chosen as horizontally and vertically symmetrical curves (similar to the available bandwidth) to add weight to the variable.

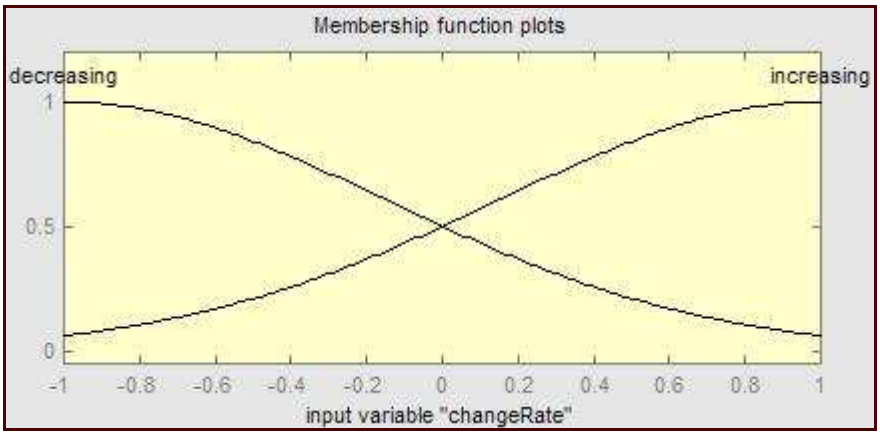

Figure 9 – The Change Rate Membership Functions.

The only output variable is the send rate. As shown in Figure 10, the send rate ranges from 0 to 1 where 0 represents a low send rate and 1 represents the maximum or high send rate. These membership functions were chosen as linear to simplify the computation and follow the design guidelines specified in [26].

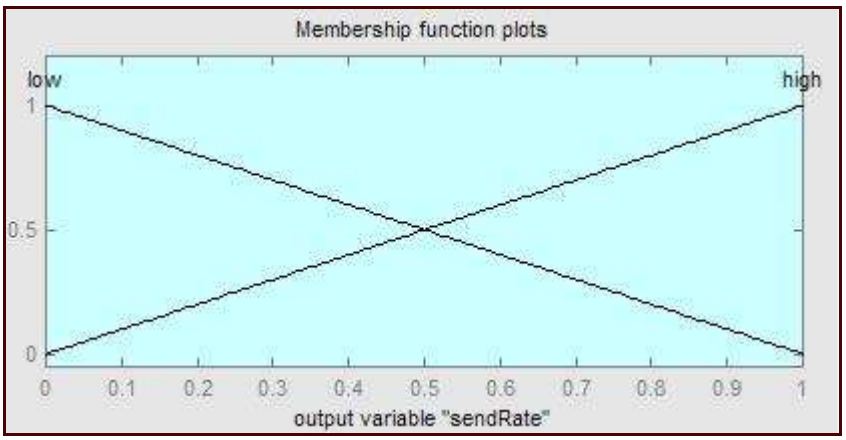

Figure 10 – The Send Rate Membership Functions.

The rules for the FIS are shown below. They utilize the AND method which is based on the min function. The FIS rules of the Fuzzy Inference System are:

1. If (changeRate is decreasing) and (availableBW is low) then (sendRate is low) (1)
2. If (changeRate is increasing) and (availableBW is low) then (sendRate is high) (1)
3. If (tcpResponse is low) then (sendRate is low) (1)
4. If (tcpResponse is high) and (availableBW is high) then (sendRate is high) (1)
5. If (availableBW is high) then (sendRate is high) (1)
6. If (availableBW is low) then (sendRate is low) (1)

For an overall picture, Figure 11 shows the MATLAB Rule Viewer. On the left side the rules are numbered 1 through 6 as above. These rows of rules apply across each column

representing the variables. The leftmost 3 columns are the inputs and below them the corresponding values chosen for each. In this example, the TCP response is .75, the change rate is 0 and the available bandwidth is .75. The rightmost column is the output and at the bottom right is the end result send rate which in this example is .636.

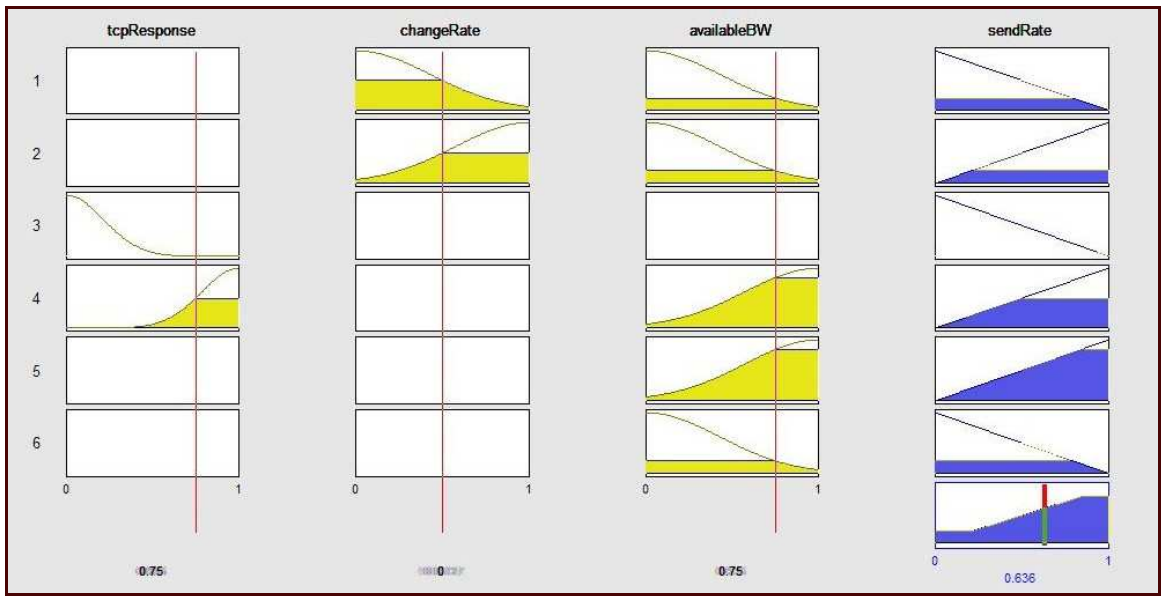

Figure 11 – Rule Viewer.

Chapter Four

Methodology and Evaluation

The solution was tested using MATLAB Simulink model simulations. Several tests were performed to demonstrate the solution's congestion control, TCP friendliness, smoothness, response time and network utilization of the proposed controller. While the stability of Fuzzy Inference Systems is still an active research area, there is no definitive analysis model but there are general design guidelines for ensuring stability as noted in [26]. The guidelines that were followed include: appropriate pre-processing (in this case normalization of the input values), overlap of membership functions to ensure well defined states, using a single output to cover the entire output range, and appropriate scaling during post-processing.

Each model environment includes an element to call the FIS, an element to emulate the available bandwidth, elements to estimate the necessary TCP inputs (RTT, PLR, packet size) and controller functions (for normalizing the inputs and doing other functions the controller will perform).

The first test, Test 1, is for smoothness, congestion collapse and utilization and shown in Figure 12. The model is a MATLAB Simulink model which represents the elements used to run the FIS controller within a created network environment and was designed to show how the TCP flow and how the controlled UDP flow would react under the

same circumstances independent of each other. The model emulates the FIS controlled UDP source and independently a TCP flow both run separately against a highly varying bandwidth changing at a rate of ±5.9 Mbps. The emulated bandwidth of the path is assumed to be 100 Mbps. Therefore, the available bandwidth can range from 0 to 100 Mbps. The packet drop rate is assumed to range from 0 to 5%. The model includes many elements to aid in emulating the network and network traffic. Starting in the upper left hand corner, the model includes a constant requested rate, which is assumed to be given by the application. The available bandwidth is generated using a MATLAB provided sine wave modified by multiplying it against a constant to vary from 0 to 100 Mbps. While this was not based on a network traffic model, it was used to demonstrate a highly varying available bandwidth. Towards the center, this input is then normalized before being submitted to the FIS. Towards the bottom left, the TCP response function is calculated. It takes input from the available bandwidth and estimates a drop rate that fluctuates with the changing available bandwidth. A low available bandwidth will produce a high drop rate and a high available bandwidth will produce a low drop rate. Using the drop rate the TCP response is calculated and then normalized for the FIS input. Towards the bottom right, the change rate is calculated as a derivative of the change in rate over unit time and fed back into the FIS as an input. Once the FIS has all the necessary inputs, it produces a send rate which is multiplied (from its 0 to 1 range) against the available bandwidth. The right hand side of the model is the graph and its inputs.

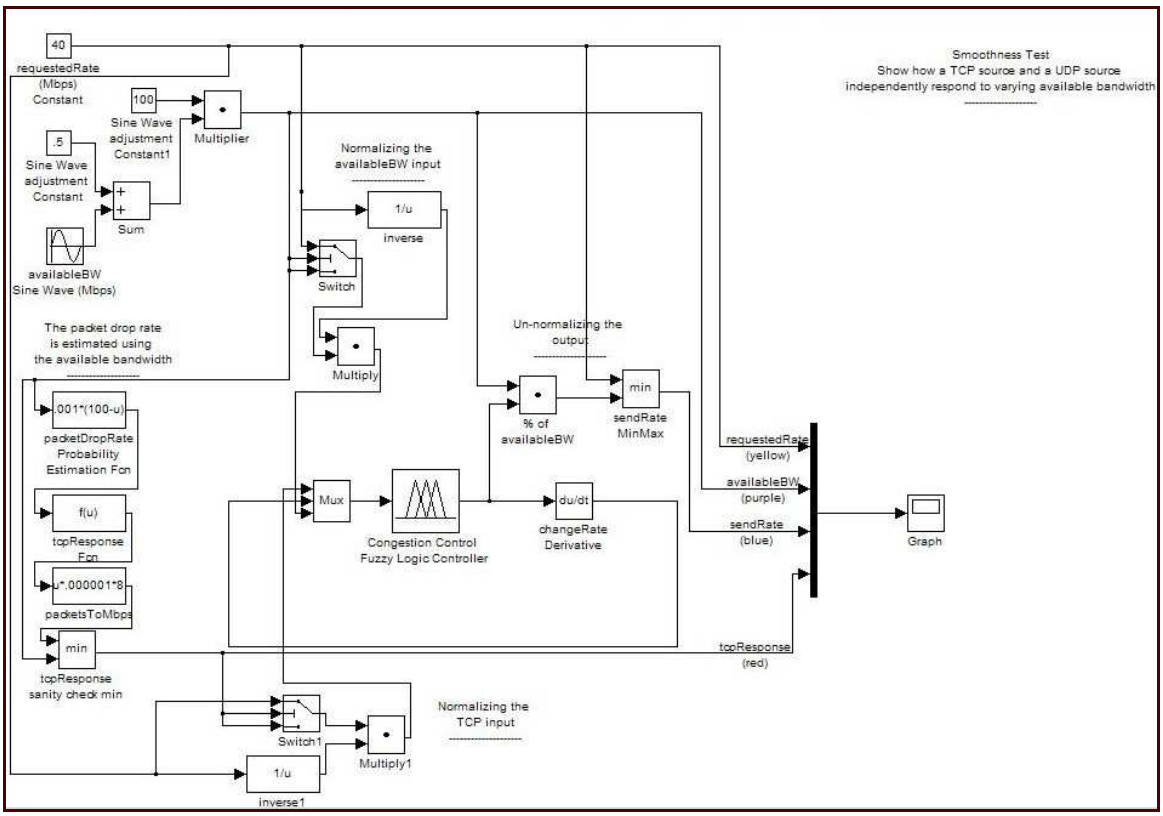

Figure 12 – Test 1 Model.

The first graph, shown in Figure 13 below comes from Test 1.

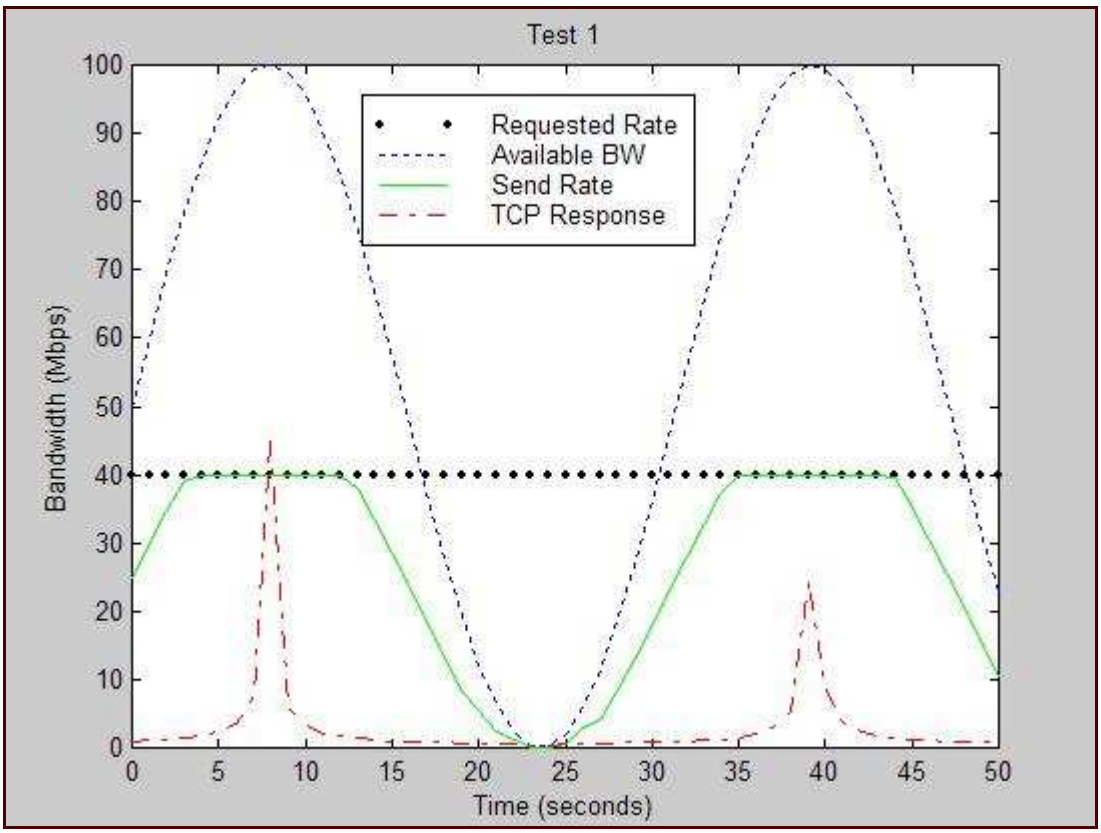

Figure 13 – Test 1 Results.

Test 1 was done to evaluate the solution's smoothness, congestion control and network utilization. Figure 13 shows the response of the controller using a rapidly varying available bandwidth input and overlaid in the graph the independently run TCP flow with the same available bandwidth. The TCP Response is slow to meet the available rate and quick to react to congestion even when there is a large amount of bandwidth available. For a streaming media application, these rate variations would have drastic consequences on the user's perceived quality. In contrast, the controlled UDP send rate is much smoother and

follows more closely the available bandwidth. It utilizes the network much more efficiently than TCP does. If the available bandwidth was 40 Mbps for 50 seconds, an ideal total of 2000 Mb could have been delivered. It is estimated that the controlled UDP delivered 1200 Mb (60% available bandwidth utilization) and the TCP Response was estimated at 320 Mb (16% available bandwidth utilization). Most importantly, the send rate is never more than the available bandwidth estimation. Therefore, if the available bandwidth estimation is accurate and timely, the solution would not insert undeliverable packets into the network and would not produce congestion collapse.

The next set of tests, Test 2a and 2b, was done for TCP friendliness. This model was designed to run the UDP and TCP flows concurrently over the same network. The emulated network is modeled after a 10 Mbps path with a 10 Mbps and a 100 Mbps link as shown in Figure 14. The drop rate can range from 0 to 5%. The available bandwidth of the path can range from 0 to 10 Mbps and the packet drop rate can range from 0 to 5%.

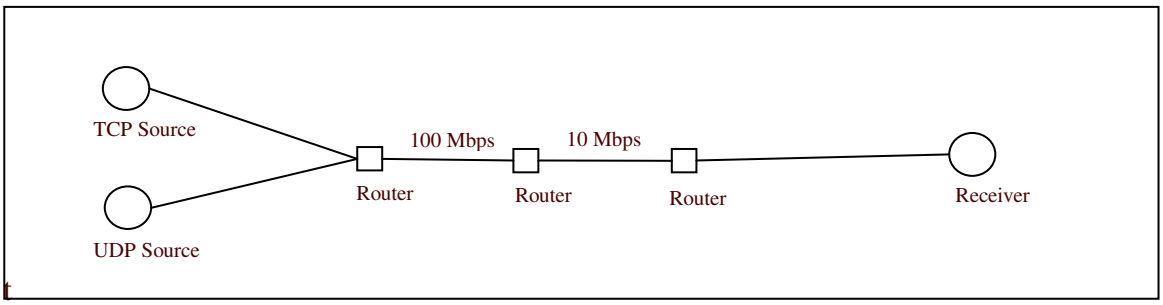

Figure 14 – Test 2 Network Configuration.

The first test in the set was done with UDP and TCP and the second was done with a controlled UDP and TCP. As shown in the Figure 15, some of the elements of the model are similar to the first test. The model includes a constant for UDP and a switch to switch UDP on after 10 seconds. The lower left section of the model still includes the TCP response. The right hand side is for the inputs and graph elements.

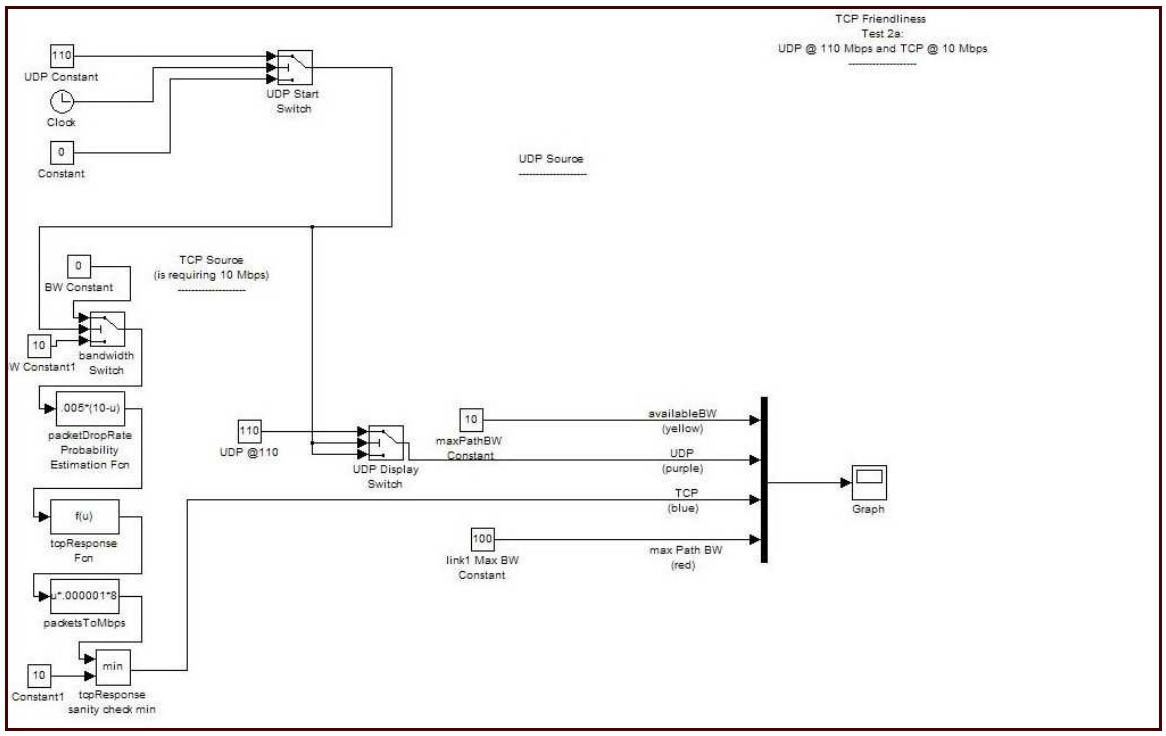

Figure 15 – Test 2a Model.

The results for the Test 2a are shown in Figure 16.

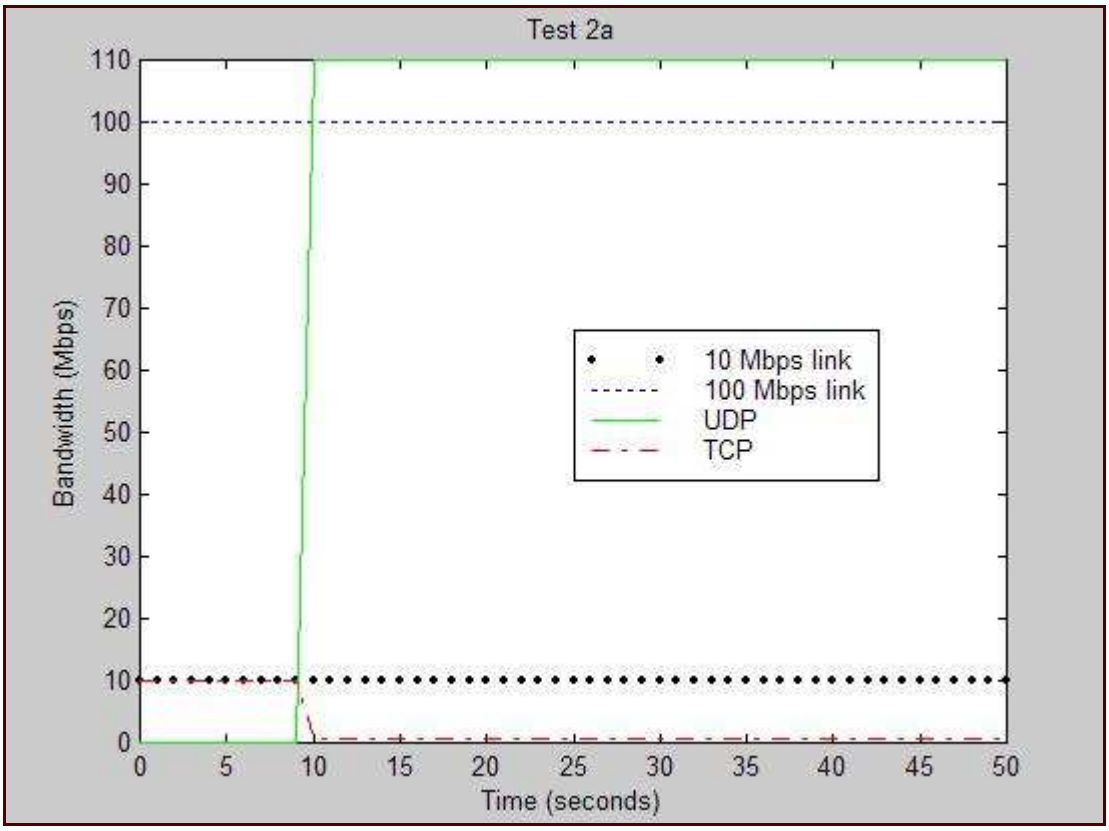

Figure 16 – Test 2a Results.

The results from Test 2a show TCP and UDP as they share a 100 Mbps and 10 Mbps link. TCP has the entire link to itself for the first 10 seconds until UDP is introduced. UDP is requesting 110 Mbps and sends at that rate unconditionally thus being extremely unfriendly towards the TCP flow. TCP reduces its rate to nothing while UDP continues to insert packets even though the network will drop most of them. The conditions can produce congestion collapse if, for example, the TCP flow was shut out from using the 100 Mbps link for a receiver off of the middle router. In that case, the UDP packets that do not reach their destination will restrict the bandwidth the TCP flow has on the first link while accomplishing no extra work.

The other test, Test 2b, in the set is modeled with similar elements to the first test except the values for available bandwidths was determined by the competing TCP and controlled UDP bandwidth usage (as opposed to a modified sine wave). The available bandwidth and TCP responses are normalized near the middle of the Figure 17 and sent through the FIS. The send rate and other outputs are sent to the graph and workspace towards the right side of the figure.

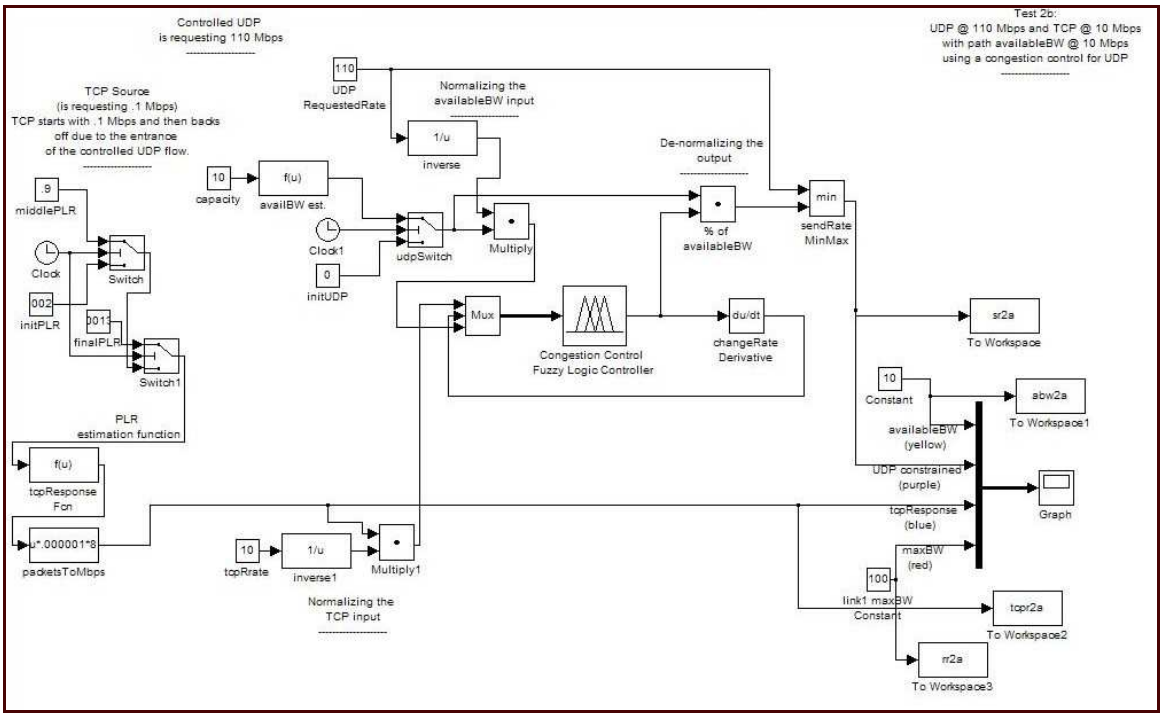

Figure 17 – Test2b Model.

The results from Test 2b are shown in Figure 18.

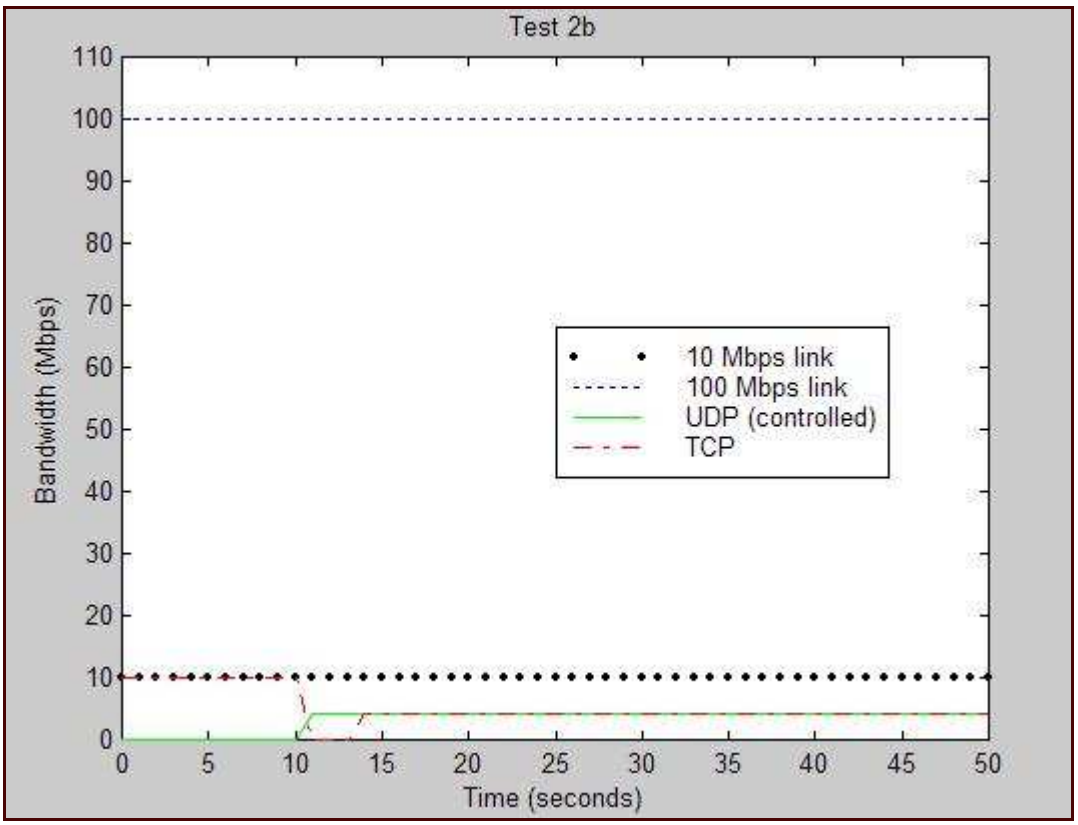

Figure 18 – Test 2b Results.

The results from Test 2b utilize the same environment as Test 2a except for using the proposed controlled UDP. The results show that no congestion collapse is produced, the number of undelivered packets is reduced by 100%, and that TCP still receives some of the limited bandwidth, nearly 50% after the initial introduction of the controlled UDP. After 10 seconds, UDP initiates its flow but does not overwhelm the network (staying under the bandwidth in the tight link in the path at less than 10 Mbps). It shares the network with the TCP flow.

The next test, Test 3, was for response time. As shown in Figure 19, the model is very similar to the first test except that the available bandwidth is dictated by workspace variables as opposed to a modified sine wave. The controlled UDP and TCP were run independently of each other under the same network conditions. The available bandwidth of the path can range from 0 to 10 Mbps and the packet drop rate can range from 0 to 5%.

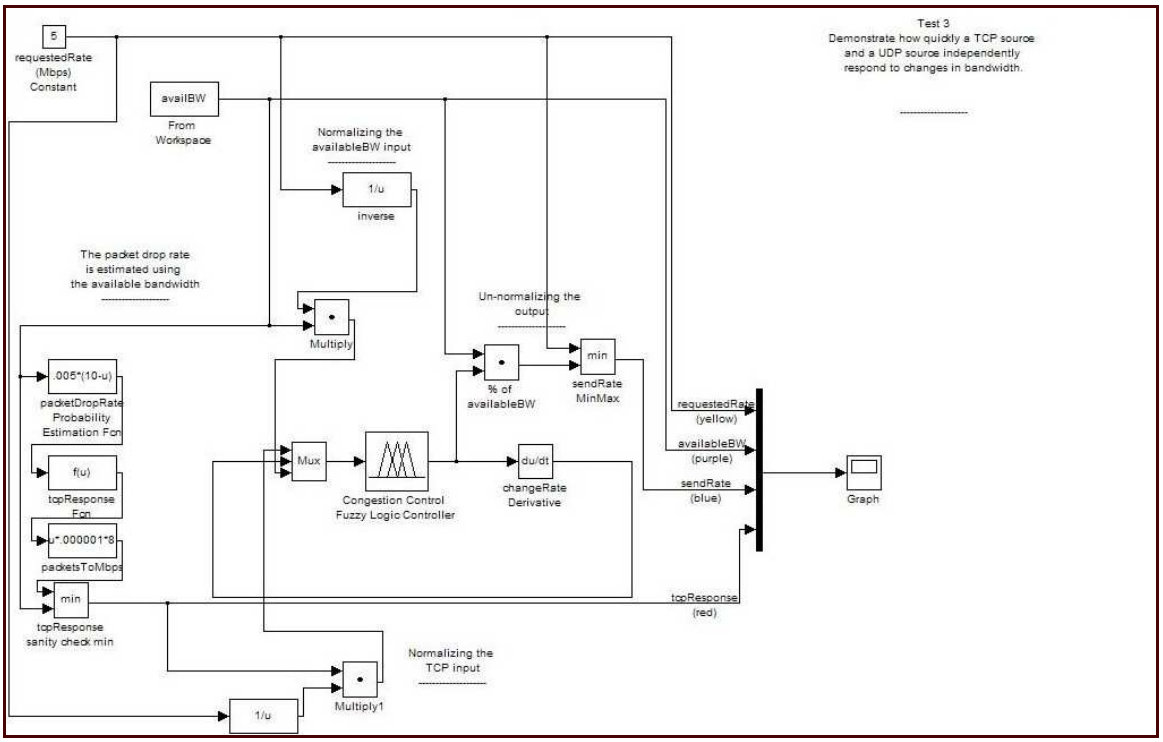

Figure 19 – Test 3 Model.

The results for the next test, Test 3, are shown in Figure 20.

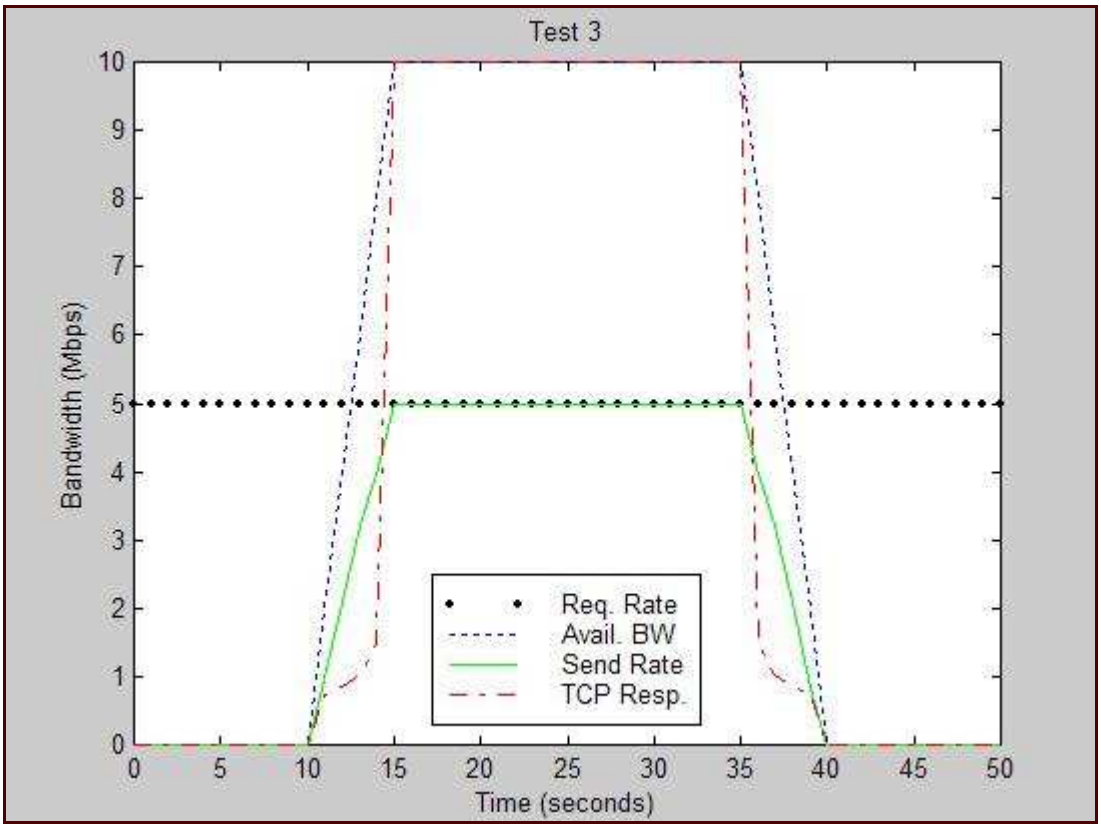

Figure 20 – Test 3 Results.

The results from Test 3 shows a requested rate of 5 Mbps and an available bandwidth that starts at 0 Mbps, at 10 seconds climbs to 10 Mbps and at 35 seconds it starts to return to 0 Mbps. The graph shows the controller send rate with the TCP response overlaid on the graph. The TCP response is quick but not necessarily smooth. The controller send rate responds quickly and relatively smoothly to both the drastic increase in available bandwidth and the rapid decrease. It also never overruns the available bandwidth or the requested rate. Also of note, the compiled C code for the FIS Controller was responding in the milliseconds CPU time range during validation testing of the code outside of the kernel.

Chapter Five

Conclusions and Future Work

In this thesis a fuzzy logic based controller is designed and evaluated to provide flow and congestion control in a transport layer protocol and make it suitable for streaming media applications. The proposed controller is meant to address other important problems in the Internet, such as the TCP-friendliness and congestion collapse problems. By means of a simulation, it is shown how the controller reduces or eliminates the possibility of congestion collapse from undelivered packets, provides an incentive for applications to use the congestion control by utilizing the network better than TCP, reacts to changes in network conditions smoother than the AIMD mechanism used by TCP, and remains friendly to TCP flows. The solution is also shown to react fast enough to accurately and appropriately respond to changes in bandwidth.

Future work for the research includes testing the solution in a simulated network or in a operating system in a live network. Using the developed C code for the controller, the solution can be integrated into a Unix system for testing. The solution should also be tested with a available bandwidth estimator to determine the optimal times to call the estimator for input and produce optimal send rates with the least impact on the network.

References

[1] Celio Albuquerque, Brett J. Vickers, and Tatsuya Suda. Network Border Patrol. In *IEEE/ACM Transactions on Networking (TON)*, Vol. 12, Issue 1, pages 173-186, February 2004.

[2] David Austerberry. *The Technology of Video and Audio Streaming: A robust technical guide to implementing an end-to-end streaming system*, pages 129-148, 2002.

[3] Hari Balakrishnan, Hariharan S. Rahul, and Srinivasan Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *ACM SIGCOMM*, September 1999.

[4] A. Balk, M. Gerla, M. Sanadidi, and D. Maggiorini. Adpative MPEG-4 Video Streaming with Bandwidth Estimation: Journal Version. In *Proc. 2nd International Workshop on QoS in multiservice IP Networks (QOS-IP 2003)*, February 24-26, 2003.

[5] Sally Floyd and Kevin Fall. Promoting the Use of End-to-End Congestion Control in the Internet. In *IEEE/ACM Transactions on Networking*, August 1999.

[6] Sally Floyd, Jitendra Padhye, and Jorg Widmer. Equation-Based Congestion Control for Unicast Applications. In *International Computer Science Institute tech report TR-00-003*, March 2000.

[7] Sally Floyd, Eddie Kohler, and Jitendra Padhye. Profile for DCCP Congestion Control ID 3: TFRC Congestion Control, draft-ietf-dccp-ccid3-05.txt, work in progress, February 2004.

[8] Yunhong Gu and Robert Grossman. Using UDP for Reliable Data Transfer over High Bandwidth-Delay Product Networks. Submitted for publication to *Computer Communication Review*.

[9] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification, RFC 3448, Proposed Standard, January 2003.

[10] Tom Kelly, Sally Floyd, and Scott Shenker. Patterns of Congestion Collapse. Under submission, June 2003.

[11] Jong Won Kim, Young-Gook Kim, HwangJun Song, Tien-Ying Kuo, Yon Jun Chung, and C.-C. Jay Kuo. TCP-Friendly Internet Video Streaming Employing Variable Frame-Rate Encoding and Interpolation. In *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 10, No. 7, October 2000.

[12] Eddie Kohler, Mark Handley, and Sally Floyd. Designing DCCP: Congestion Control Without Reliability. Submitted to *International Conference on Network Protocols*, 2003.

[13] Eddie Kohler, Mark Handley, and Sally Floyd. Datagram Congestion Control Protocol (DCCP). draft-ietf-dccp-spec-06.txt, work in progress, February 2004.

[14] L.-A. Larzon, M. Degermark, and S. Pink. UDP Lite for Real Time Multimedia Applications. *Hewlett Packard Reports HPL-IRI-1999-001*. April 1999.

[15] Qiang Liu and Jenq-Neng Hwang. End-toEnd Available Bandwidth Estimation and Time Measurement Adjustment for Multimedia QOS. In *IEEE Internation Conference on Multimedia and Expo*, Vol. III, pages 373-376, 2003.

[16] S. McCanne and S. Floyd. NS Network Simulator. http://www.isi.edu/nsnam/ns.

[17] J. Postel. User Datagram Protocol. Internet Request For Comments RFC 768. August 1980.

[18] Ravi Prasad, Constantinos Dovrolis, Margaret Murray, and KC Klaffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. In IEEE Network, pages 27-35, November/December 2003.

[19] Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya-Qin Zhang, and H. Jonathan Chao. MPEG-4 Compressed Video over the Internet. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS'99)*, May 30-June 2, 1999.

[20] J. Widmer, R. Denda, and M. Mauve. A Survey on TCP-friendly congestion control. In *IEEE Mag. Network,* Vol. 15, pp. 28-37, May/June 2001.

[21] Fan Yang, Qian Zhang, Wenwu Zhu, Ya-Qin Zhang. End-toEnd TCP-Friendly Streaming Protocol and Bit Allocation for Scalable Video Over Wireless Internet. In *IEEE Journal on Selected Areas in Communications*, Vol. 22, No. 4, May 2004.

[22] Qian Zhang, Wenwu Zhu, and Ya-Qin Zhang. Network-adaptive Rate Control with TCP_Friendly Protocol for Multiple Video Objects. In *IEEE International Conference on Multimedia and Expo (ICME)*, July 2000.

[23] Wenwu Zhu, Qian Zhang, and Ya-Qin Zhang. Network-Adaptive Rate Control with Unequal Loss Protection for Scalable Video over Internet. In *IEEE International Symposium on Circuits and Systems*, pages 109-112, 2001.

[24] J. Postel. Transmission Control Protocol. Internet Request For Comments RFC 793. September 1981.

[25] Jerry M. Mendel. Fuzzy Logic Systems for Engineering: A Tutorial. In *Proceedings of the IEEE*, pages 345-356, Vol. 83, No. 3, March 1995.

[26] Jan Jantzen. Design of Fuzzy Controllers, *Tech. report no 98-E 864 (design),* 1998.

[27] Bernd Girod, Mark Kalman, Yi J. Liang, Rui Zhang. Advances in Channel-Adaptive Video Streaming. In *Proc. IEEE Intern. Conf. Image Processing*, Sept. 2002.

[28] Lotfi A. Zadeh. Fuzzy sets, *Inf. Control 8*, pages 338-353, 1965.

[29] S. Ghosh, Q. Razouqi, H.J. Schumacher, and A. Celmins. A Survey of Recent Advances in Fuzzy Logic in Telecommunications Networks and New Challenges. In *IEEE Transactions on Fuzzy Systems*, Vol. 6, No. 3, August 1998.

[30] C. Douligeris and G. Develekos. A Fuzzy Logic Approach to Congestion Control in ATM Networks. In *IEEE ICC*, 1995.

[31] D. Miras and G. Knight. Smooth Quality Streaming of Live Internet Video. In *IEEE GLOBECOM*, 2004.