FUZZY END-TO-END RATE CONTROL FOR INTERNET TRANSPORT PROTOCOLS

F. Montesino[1], D. R. López[2], A. Barriga[1], S. Sánchez-Solano[1]

[1] Instituto de Microelectrónica de Sevilla - Centro Nacional de Microelectrónica
Avda. Reina Mercedes s/n, (Edif. CICA), E-41012, Sevilla, Spain.

[2] RedIRIS, Spanish NREN. Edif. Bronce,
Pza. Manuel Gómez Moreno s/n. Planta 2. E-28020 Madrid, Spain

Proc. IEEE International Conference on Fuzzy Systems (FUZZIEEE 2006),

Vancouver, July 16-21, 2006.

Fuzzy End-to-End Rate Control for Internet Transport Protocols

Federico Montesino, Diego R. Lopez, Ángel Barriga and Santiago Sánchez-Solano

Abstract— **End-to-end Internet packet dynamics is a complex problem for which models available to date are at best incomplete. A major research problem in Internet transport layer protocols is the development of rate control mechanisms that can cope with the requirements of a growing diversity of technologies, applications and services. This paper describes novel mechanisms for intelligent end-to-end traffic rate control in Internet by means of fuzzy systems. We first outline a fuzzy logic based generalization of TCP (Transport Control Protocol) rate control principles. The design of a fuzzy TCP-like window-based rate controller is then described. A systematic fuzzy systems design methodology is used in order to simulate and implement the system as an experimental tool. A comparative evaluation of simulation an implementation results from the fuzzy rate controller as compared to that of traditional controllers is outlined. Besides being a useful modelling approach, the fuzzy rule based rate controller is shown to outperform other approaches with regards to a number of criteria.**

I. INTRODUCTION

End-to-end Internet packet dynamics is a complex problem for which models available to date are at best incomplete [1]. A great deal of attention is being payed by the research community to two issues in this area: protocols for high performance networking, and solutions for new services and applications for which proper congestion control schemes are sought. These issues have lead to an interest in the application of artificial intelligence techniques to end-to-end problems within the End-to-End Research Group of the Internet Research Task Force.

As shown in figure 1, currently deployed schemes for traffic regulation in Internet (as well as proposed alternatives) fit into one of the two following approaches [2]:

- Distributed control, with functionality distributed among the end nodes in the network and implemented by means of end-to-end transport protocols. Transmitter and receiver end nodes of packet flows cooperate so as to perform flow and congestion control as well as fair distribution of network resources.
- Queue schedulers in intermediate nodes (routers). These mechanisms can discriminate packet flows and enforce resource distribution and reservation.

Thus, regulation of packet flows from sender to receivers can involve all the network nodes in the end-to-end path and is performed on both an end-to-end and a per-hop basis. Such a scheme leads to a system that comprises multiple feedback loops with complex interactions.

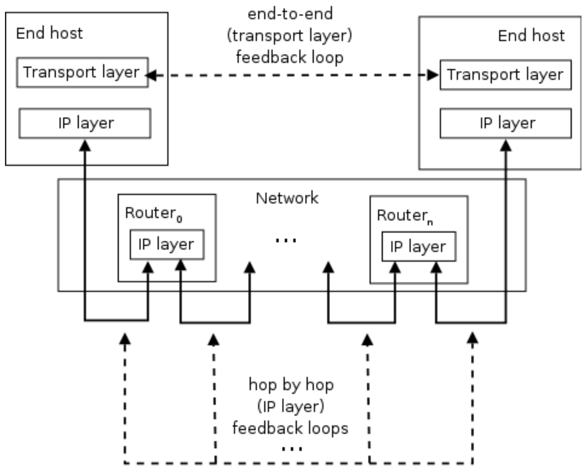

Fig. 1
FEEDBACK LOOPS IN INTERNET TRAFFIC REGULATION

End-to-end flows traversing routers span a wide range of user requirements and dynamic characteristics, i.e., the number of hops in the path can tipically range from a few up to around 20, round-trip times can range from a few milliseconds up to seconds, flows can last from a few milliseconds up to hours and each flow can transfer from a few KBs up to several GBs.

Quality of service requirements as well as traffic patterns of emergent services and applications are difficult to characterize and demand deep advances in current rate control schemes. Because of the nature of these problems (complexity, no feasible analytic solution as well as incomplete and inaccurate information) one of the alternatives for studying them is the employment of intelligent systems based on fuzzy logic and possibly other soft computing techniques.

Both aforementioned approaches can be redefined in terms of fuzzy systems, which does not only provide a deeply backgrounded engineering approach but also a modelling and analysis framework for Internet traffic (which the current Internet research community lacks [1]). In this paper we focus on end-to-end rate control. More specifically, we analyze TCP-like window-based rate control. With this work we aim at providing a reinterpretation of end-to-end flow and congestion control mechanisms in terms of fuzzy systems. We provide an initial set of results based on simulation and experimental implementation showing a number of advantages as for both performance and model interpretability.

Section II provides an overview of related publications.

Federico Montesino Pouzols, Angel Barriga Barros and Santiago Sánchez-Solano are with the Microelectronics Institute of Seville, Spanish Scientific Research Council, Avda. Reina Mercedes s/n. Edif. CICA. E-41012 Seville, Spain (phone: +34 955 056 666; fax: +34 955 056 686; email: {fedemp,barriga,santiago}@imse.cnm.es).

Diego R. Lopez is at RedIRIS, the Spanish National Research and Education Network, Edificio Bronce, Pza. Manuel Gómez Moreno s/n. Planta 2. E-28020 Madrid, Spain (phone: +34 912 127 625; fax: +34 915 568 864; email: diego.lopez@rediris.es)

Section III outlines a fuzzy logic based extension to end-to-end window based rate control schemes. In section IV we detail the design of a fuzzy system for end-to-end rate control. Section V gives an overview of the fuzzy systems development methodology and tool chain employed. Simulation and experimental implementation results are then presented in section VI and section VII, respectively. Finally, we conclude in section VIII.

II. Related Work

Formal approaches that have been applied to end-to-end rate control include classic control theory and fluid mechanics among others [2]. While a number of proposals of fuzzy controllers for packet queues at Internet routers have been reported, see [3], [4], among others, the only reference we are aware of that takes a fuzzy logic based approach to end-to-end rate control is [5], which outlines a perliminary study through an off-line simulation based analysis of a non-linear Takagi-Sugeno fuzzy controller applied to some of the basic mechanisms of TCP congestion control.

In the area of wireless networks, there have been proposals of fuzzy systems applied to intelligent discrimination of congestion induced packet loss and loss due to physical channel errors and mobility [6], [7].

III. End-to-End Window Based Rate Control and a Fuzzy Generalization

A number of classes of rate control have been defined and applied in packet switching networks, such as window based and equation based [1]. The prevalent transport protocol in the current Internet is Transmission Control Protocol (TCP), which uses window based rate control mechanisms.

TCP includes basic mechanisms for flow control since its original specification was published. After a series of congestion collapses in the network [8], mechanisms for congestion control have been added throughout the years. The first proposal of the now widely accepted congestion avoidance algorithm was introduced [8], standardized anf further developed [9]. This process lead to the development, standardization and wold-wide deployment of congestion control mechanisms that together with the basic flow control mechanisms comprise the core rate control functional block of TCP. TCP rate control comprises four intertwined algorithms: slow start, congestion avoidance, fast retransmit and fast recovery.

Let us consider the simplified version in algorithm 1 (of historical [8] interest only) for the sake of simplicity in explaining our approach. W_{max} is the delay-bandwidth product of the network path. Details of current standard algorithms and how congestion is detected (commonly based on packet loss) are provided in the next section.

The algorithm tries to react quickly to congestion conditions. When the network is congested, the amount of traffic from competing flows must be large and the queue lengths will start increasing exponentially. Under the assumption that the system will stabilize if the traffic sources throttle back at least as quickly as the queues are growing and considering

Algorithm 1 Basic AIMD

```
if congestion then
    $W_i = d\,W_{i-1}, \quad (d < 1)$
else
    $W_i = W_{i-1} + u, \quad (u \ll W_{max})$
end if
```

that a source controls load in a window-based protocol by adjusting the size of the window, W, we end up with the sender policy $W_i = d\,W_{i-1}$, i.e. a multiplicative decrease rate that under persistent congestion leads to an exponential decrease of the sender window.

If there is no congestion, router queues must be near zero and the network load approximately constant. The network announces, via a dropped packet, when demand is excessive but does not notify if a connection is using less than its fair share (since the network is stateless, this information is not available). Thus a connection has to increase its bandwidth utilization to find out the current upper limit. The first thought is to use a multiplicative increase rate possibly with a longer time constant, $W_i = b\,W_{i-1}, 1 < b < 1/d$. This is however a mistake. The result will wild oscillations and poor average throughput. The analytic reason for this, well known in queueing systems as the rush-hour effect, is due to the fact that it is easy to drive the net into saturation but hard for the net to recover. An increase policy based on small, constant changes was proposed for TCP [8]. This policy has proven to be effective.

Nonetheless, the standard additive increase multiplicative decrease (AIMD) scheme can be too cautious under some conditions, which has given rise to a great deal of research towards protocols for high-perforamce networks. A number of important limitations in TCP rate control are currently recognized and the Internet research community is developing protocols with alternative rate control schemes, such as the TCP-Friendly Rate Control protocol (TFRC) [10], and protocols with similar yet extended schemes for rate control, such as the Stream Control Transmission Protocol (SCTP) [11]. A number of extensions and modifications to TCP rate control are being developed as well, such as HighSpeed TCP [12], FAST TCP [2], H-TCP [13], and Scalable TCP [14].

As stated above, interactions between network layers, end and intermediate nodes lead to a complex non-linear dynamic system that makes it difficult to develop, simulate, and test rate control schemes. A great deal of theoretical analysis are currently being performed and a number of experimental implementations have been proposed. In this work we address the problem by means of fuzzy systems. In what follows we describe a fuzzy model for TCP-like rate control in an incremental manner.

In the simplified algorithm above, we distinguish two window evolution stages. Each stage corresponds to a clearly identified network state and leads to the application of a specific window update policy. However, the actual state can be in between these two crisp conditions. Additionaly the knowledge about network state is uncertain and delayed. We note here a binary logic problem: when the network state for which the rate control stage has been defined is constant for enough time, the system response is proper. However, when the network state does not exactly match any of the stages but a combination of them, the response of the system may be too aggressive or too conservative.

Similarly to the proposal in [5], a first generalization of the algorithm above could be stated by means of a simple reformulation of the basic AIMD principle:

$$w_{i+1} = w_i + \alpha_D f_D(w_i) + \alpha_I f_I(w_i) \tag{1}$$

Where f_D and f_I set the increase and decrease policies. α_D and α_I can be thought as degree of truth values that represent to what extend the system is on the congested (window decrease) or uncongested (window increase) mode; these values can be defined as mutually exclusive, $\alpha_D, \alpha_I \in 0, 1; \alpha_D = \overline{\alpha_I}$. This formulation suggests a fuzzy approach for managing the window update process. Instead of considering the network in one of a set of exclusive states, we will consider the network to be (to a variable degree) in all defined states. The degree to what the network is considered to be in a particular state will be identified by a fuzzy rule based inference system.

IV. Design of a Fuzzy End-to-End Window-Based Rate Controller

It is out of the scope of this paper to provide a complete description of the TCP rate control and related algorithms. We will focus on those procedures that perform a direct action on the window, i.e. those procedures that imply the definition of a network state (and a stage in the rate control algorithm) as well as the associated window update policy. For a full specification of TCP, please, refer to the aforementioned documents. Though our approach finds applications in general window-based end-to-end transport protocols, we analyze the case of TCP as a case of special interest. On what follows we will describe in an incremental manner those mechanisms defined for rate control in standard TCP implementations and how we have extended them by means of fuzzy inference systems.

The four standard algorithms for controlling the congestion window in standard TCP systems (slow start, congestion avoidance, fast retransmit and fast recovery) will be described. On what follows, definitions in table I are considered.

TABLE I
TCP Rate Control Parameters

Parameter	Description
$SMSS$	Sender Maximum Segment Size. Size of the largest segment that the sender can transmit
$rwnd$	The most recently advertised receiver window
$cwnd$	Congestion window. A limit on the amount of data TCP can send. At any given time, TCP does not send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of $cwnd$ and $rwnd$.
$flightSize$	The amount of data that has been sent but not yet acknowledged.
RTT	Round-trip time
RTO	Retransmission timeout (which depends on RTT
IW	Initial window value for cwnd
LW	Loss window value
$ssthresh$	Threshold between Slow start and Congestion avoidance stages

A. Slow Start

Beginning transmission into a network with initial unknown conditions requires TCP to cautiously probe the network to estimate the available capacity, in order to avoid congesting the network with an inappropriately large burst of data. The slow start algorithm is used for this purpose at the beginning of a transfer. The initial $cwnd$ value is usually a few $SMSS$ (most common value is $2SMSS$).

The initial value of $ssthresh$ may be arbitrarily high (usually the size of the receiver advertised window) though may be reduced in response to congestion. $ssthresh$ is used to select which window update policy should be applied. As specified in the standard, listed in algorithm 2:

Algorithm 2 Standard TCP AIMD
```
if $cwnd < ssthresh$ then
    Perform slow start
else if $cwnd > ssthresh$ then
    Perform congestion avoidance
else
    Perform either slow start or congestion avoidance
end if
```

In the standard TCP slow start stage, $cwnd$ is incremented each time an acknowledgement packet is received from the sender that acknowledges new data[1] The increment rate is defined as follows:

$$cwnd_{i+1} = cwnd_i + inc, \quad (inc \leq SMSS) \tag{2}$$

It is common for current implementations to choose inc = $SMSS$. The slow start stage (with exponential growth with time in the window) ends when $cwnd$ is greater than (or greater or equal to) $ssthresh$ or when congestion is observed.

As a fuzzy extension to the slow start algorithm we propose an inference system (*SlowStartConfidende*) that produces as output the extent to which current network conditions should be managed with the slow start algorithm. In other words, the system infers an estimate of certainty about the network being in such a state that should be handled

[1]This point is currently under revision, as better performance can be achieved by counting the number of octets (instead of the number of packets) acknowledged.

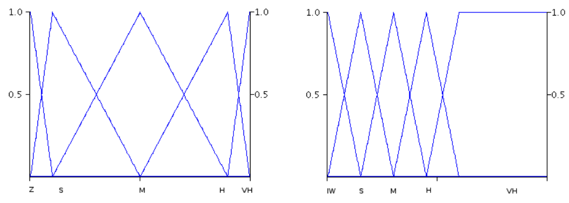

Fig. 2

$Ttimeout$ AND $Tcwnd$ MEMBERSHIP FUNCTIONS

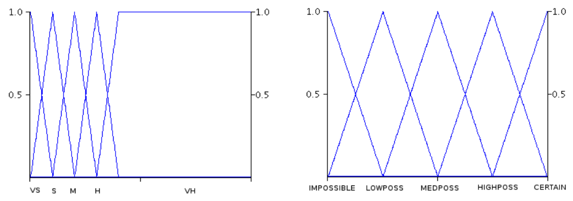

Fig. 3

$Tloss$ AND $Tconfidence$ MEMBERSHIP FUNCTIONS

TABLE II

SLOWSTARTCONFIDENCE RULE BASE.

$timeout$	$cwnd$	$ssthresh$	$ss - cw$	$loss$	rtt	$conf.$
Z	x	x	x	x	x	HI
S	x	x	x	x	x	HI
M	x	x	x	x	x	ME
H	x	x	x	x	x	ME
VH	x	x	x	x	x	IM
x	IW	x	x	x	x	CE
x	S	x	x	x	x	LO
x	M	x	x	x	x	LO
x	H	x	x	x	x	LO
x	VH	x	x	x	x	IM
x	x	VS	x	x	x	IM
x	x	S	x	x	x	LO
x	x	M	x	x	x	LO
x	x	H	x	x	x	LO
x	x	VH	x	x	x	ME
x	x	x	VS	x	x	IM
x	x	x	S	x	x	LO
x	x	x	M	x	x	ME
x	x	x	H	x	x	HI
x	x	x	VH	x	x	CE
x	x	x	x	VH	VH	IM
x	x	x	x	VH	H	LO
x	x	x	x	H	VH	LO
x	x	x	x	VS	VS	ME

by means of the slow start update policies. The system has the following inputs: $timeout$ (to which extent a timeout is expiring), $ssthresh$, $cwnd$, the difference $ssthresh - cwnd$, and two inputs that provide information on overall network conditions: the cummulative packet loss fraction, $loss$, and the round-trip time, rtt.

Figure 2 shows the fuzzy types and linguistic terms for inputs $timeout$ and $cwnd$ ($Ttimeout$ and $Tcwnd$). Figure 3 shows the types for input $loss$ and the output, $confidence$. In order to simplify definition and easing the use of efficient implementation techniques, only triangular and trapezoidal membership functions are used. In general, the fuzzy types are defined using a partition of the crisp input space and triangular and trapezoidal membership functions. 5 linguistic terms are defined for inputs, and 5 linguistic terms are defined (for increasing degrees of certainty from IMPOSSIBLE to CERTAIN) for the output.

Fuzzy inference follows the Mamdani model, and the fuzzy mean defuzzification method is employed to compute the crisp output value. Membership functions were adjusted considering typical performance values considered in recent Internet measurement studies [15].

Table II shows the rule base of the *SlowStartConfidende* system. Standard TCP rate control activates slow start update policies when $cwnd$ is lower than $ssthresh$. As can be seen in the rule base, the fuzzy system yields a certainty degree that increases with input $ssthresh - cwnd$ (fourth column). This behavior can be thought as a generalization of the crips lower than comparison used in standard TCP. The five fuzzy rules triggered by input $ssthresh - cwnd$ represent the informal expression "the bigger $ssthresh - cwnd$, the more possibility of slow start being suitable for current network conditions".

However, additional rules can modify the certainty degree if other network conditions suggest a different update policy. Rules that most directly reproduce the non-fuzzy standart TCP behavior are those triggered by values of the $ssthresh - cwnd$ input. The degree of certainty given by these five rules is then ajusted by additional rules that consider further information on current network conditions. Four rules are at least fired at any given time. Exactly one of the first five rules (first five rows) is always active (for input $timeout$), as well as exactly one of the next four rules (for input $cwnd$). The same applies to $ssthresh$, and $ssthresh - cwnd$. Rules depending on $loss$ and rtt are triggered only under clear network saturation conditions.

B. Congestion Avoidance

In the congestion avoidance stage, $cwnd$ is incremented by $SMSS$ per round-trip time. In real implementations, it is common to increment $cwnd$ for every non duplicated acknowledgement packet received from the receiver as in:

$$cwnd_{i+1} = cwnd_i + SMSS \cdot SMSS/cwnd_i \qquad (3)$$

Which is generally considered to be an acceptable approximation and leads to a growth rate linear with time. Congestion avoidance ends when congestion is detected.

In addition, in any stage $ssthresh$ and $cwnd$ are modified when a timeout is detected according to:

$$ssthresh = max(flighSize/2, 2\,SMSS) \qquad (4)$$

$$cwnd_{i+1} = cwnd_{TO} \leq LW \qquad (5)$$

TABLE III
CongestionAvoidanceConfidence Rule Base.

$timeout$	$cwnd$	$ssthresh$	$ss-cw$	$loss$	rtt	$conf.$
Z	x	x	x	x	x	IM
S	x	x	x	x	x	IM
M	x	x	x	x	x	IM
H	x	x	x	x	x	ME
VH	x	x	x	x	x	CE
x	IW	x	x	x	x	IM
x	S	x	x	x	x	LO
x	M	x	x	x	x	LO
x	H	x	x	x	x	LO
x	VH	x	x	x	x	HI
x	x	VS	x	x	x	CE
x	x	S	x	x	x	LO
x	x	M	x	x	x	LO
x	x	H	x	x	x	IM
x	x	VH	x	x	x	IM
x	x	x	VS	x	x	CE
x	x	x	S	x	x	HI
x	x	x	M	x	x	ME
x	x	x	H	x	x	LO
x	x	x	VH	x	x	IM
x	x	x	x	VH	VH	CE
x	x	x	x	VH	H	HI
x	x	x	x	H	VH	HI
x	x	x	x	H	H	ME
x	x	x	x	M	VH	ME
x	x	x	x	M	H	LO

With LW usually being set to 1 full-sized segment. As with the slow start algorithm, we propose an extension to congestion avoidance. The extension consists of a fuzzy inference system (*CongestionAvoidanceConfidence*) that produces as output the extent to which current network state should be managed following the congestion avoidance algorithm. The system inputs are the same as those of *SlowStartConfidence*.

Table III shows the rule base, which has a similar structure to that of *SlowStartConfidence* but employs what can broadly be seen as a complementary rule base. Note however that the rules depending on $loss$ and rtt are triggered under different conditions. In this case, the five rules triggered by $ssthresh - cwnd$ represent the following sentence: "the smaller $ssthresh - cwnd$, the more possibility of congestion avoidance being a proper algorithm for current network conditions".

C. Fast Retransmit and Fast Recovery

TCP receivers send an immediate duplicate acknowledgement (ACK) packet back to the sender when an out-of-order segment arrives. The purpose of this ACK is to inform the sender that a segment was received out-of-order and which sequence number was expected instead. From the sender's perspective, duplicate ACKs can be caused by a number of network problems. They can be caused by dropped segments. In this case, all segments after the dropped segment will trigger duplicate ACKs. In addition, duplicate ACKs can be caused by the re-ordering of data segments by the network. Finally, duplicate ACKs can be caused by replication of ACK or data segments by the network.

TCP senders use the *fast retransmit* algorithm to detect and repair loss, based on incoming duplicate ACKs. The fast retransmit algorithm uses the arrival of 3 duplicate ACKs After receiving 3 duplicate ACKs, TCP senders perform a retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire. After the fast retransmit algorithm sends the apparently missing segment, the *fast recovery* algorithm governs the transmission of new data until a non-duplicate ACK arrives. These two algorithms are usually implemented together as follows.

1) When the third duplicate ACK is received, the loss segment is retransmitted and $ssthresh$ and $cwnd$ are set to

$$ssthresh = max(flightSize/2, 2\,SMSS) \tag{6}$$

$$cwnd_{i+1} = cwnd_{DUP3} = ssthresh + 3\,SMSS \tag{7}$$

This artificially inflates the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.

2) For each additional duplicate ACK received, increment

$$cwnd_{i+1} = cwnd_i + SMSS \tag{8}$$

This artificially inflates the congestion window in order to reflect the additional segment that has left the network.

3) Transmit a segment, if allowed by the new value of $cwnd$ and the receiver's advertised window.
4) When the next ACK arrives that acknowledges new data, set $cwnd$ to $ssthresh$ (the value set in step 1).

$$cwnd_{i+1} = ssthresh \tag{9}$$

As with the slow start and congestion avoidance algorithms, we propose an extension that consists of a fuzzy inference system (*FRFRConfidence*) that produces as output the extent to which the fast recovery/fast retransmit strategy is suitable for current network conditions. The inputs to the system are $timeout$, $ssthresh - cwnd$, $loss$ and rtt. The rule base is shown in table IV. Exactly one of the first five rules (first rows) will be triggered at any time for different values of $timeout$. The same applies to the next five rules (for $ssthresh - cwnd$ values). The last four rules are triggered when network conditions suggest that the fast recovery, fast retransmit update policies are suitable.

D. Putting All Pieces Together

In our extended approach, we define three fuzzy stages (for slow start, congestion avoidance and fast retransmit/fast recovery). Since the actual network state is known with uncertainty, all network states are considered to occur at the same time with a varying degree of certainty. The extent to which the system is in one of these stages is evaluated by three fuzzy inference systems whose outputs are regarded as a degree of certainty about current network state. This way, we model uncertainty about the actual network congestion state.

TABLE IV

FRFRConfidence Rule Base.

$timeout$	$ssthresh - cwnd$	$loss$	rtt	$conf.$
Z	x	x	x	IM
S	x	x	x	IM
M	x	x	x	IM
H	x	x	x	IM
VH	x	x	x	CE
x	VS	x	x	IM
x	S	x	x	LO
x	M	x	x	ME
x	H	x	x	HI
x	VH	x	x	CE
x	x	VH	VH	CE
x	x	VH	H	HI
x	x	H	VH	LO
x	x	M	H	HI

Update policies of $ssthresh$ and $cwnd$ given in equations 2 to 9 are kept. However, as the network state is in general uncertain, all policies are simultaneously evaluated and applied to a varying degree given by the three fuzzy inference systems described. Thus, the three inference systems identify complex network states on the basis of linguistic rules.

Slow start and congestion avoidance equations are always evaluated as in standard TCP implementations (equations 2 to 5). Additionally, when duplicated ACKs are detected, fast retransmit and fast recovery (equations 6 to 9) are evaluated as well (and considered to the extent given by the *FRFRConficence* fuzzy inference system).

In the simplest case, the three set of policies are combined as follows. If we denote by μ_{ss}, μ_{ca}, and μ_{FRFR} the output of the three fuzzy inference systems, and by f_{SS}, f_{CA} and f_{FRFR} the values given by the standard update policies (which we will generalize to μ_i and f_i ($1 \leq i \leq n$) for a variable number n of network states (or stages in the rate control algorithm, or sets of update policies)), we define the certainty degree c_i of a stage as:

$$c_i = \frac{\mu_i}{\sum_{j=1}^{n} \mu_j}$$

Where the rule sets of the fuzzy inference systems should verify $\sum_{j=1}^{n} \mu_j \geq 0$. The final update policy for $cwnd$ is computed as the weighted average of the update policies associated to all possible stages (as in equation 1):

$$cwnd_{i+1} = cwnd_i + \sum_{i=1}^{N} c_i f_i(cwnd, ssthresh)$$

This way, we basically have a rule based method of combining update policies that have been shown to be effective under certain conditions. The rule sets can lead to compromise solutions under complex conditions. For instance, policy 9 usually implies a large decrease of the congestion window. When additional inputs indicate network congestion, the policy may be adequate. However, when the network does not appear to be congested, a more aggressive behavior could improve performance in terms of throughput and responsiveness.

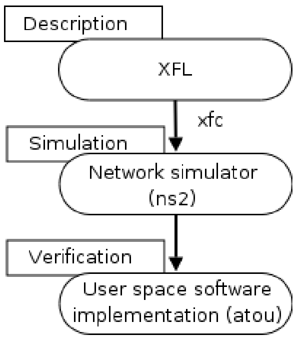

Fig. 4

Fuzzy Systems Design Flow and Tool Chain

V. Development Methodology and Tool Chain

In order to develop fuzzy inference systems, we adhere to a design methodology [16] for the whole development process that covers from initial high-level description to implementation as software and hardware components. A complete tool chain for the development stages [17] has been employed.

As a result from more than a decade of research experience on the digital implementation of fuzzy systems, the fuzzy group at IMSE has developed methodologies and CAD tools that fulfill the design flow of fuzzy systems. Leveraging on the Xfuzzy [17] CAD suite of tools and a methodology [16] for the development of fuzzy controllers, we have defined a methodology and tool chain tailored for the development of fuzzy Internet rate controllers.

The design flow and tool chain employed to develop fuzzy inference modules is depicted in figure 4. The whole development process is covered, from initial description to final implementation whether as software or hardware. The first development stage (description) is performed using a high level fuzzy systems specification language, XFL [18], which can be automatically turned into C and VHDL code among other implementation options.

The development stages after specification have been tailored for end-to-end rate control as follows. For network simulation, we have used ns-2 [19]. ns-2 is an object oriented discrete event driven simulator with support for a vast variety of transport protocols, queueing systems, routing schemes and access media, thus enabling us to evaluate the performance of traffic controllers under complex and realistic simulated scenarios. Fuzzy controllers are integrated into ns-2 as components implemented in C.

Verification can be performed over software and hardware implementations of fuzzy controllers. Software verification is performed over a controller implementation within a tool that implements the TCP protocol in user space (atou), which is further described in section VII.

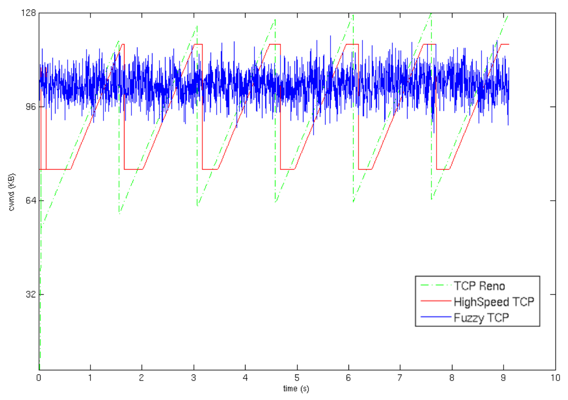

Fig. 5

CONGESTION WINDOW EVOLUTION (SIMULATION)

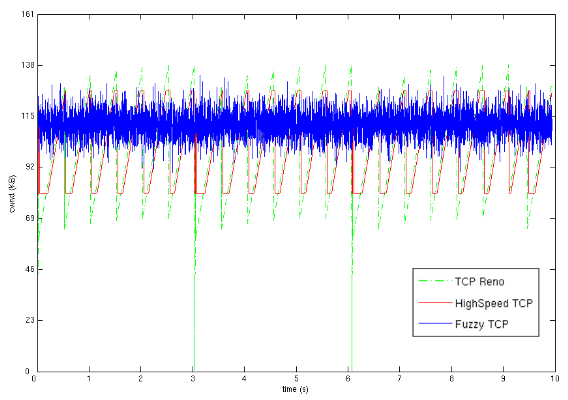

Fig. 7

CONGESTION WINDOW EVOLUTION (IMPLEMENTATION)

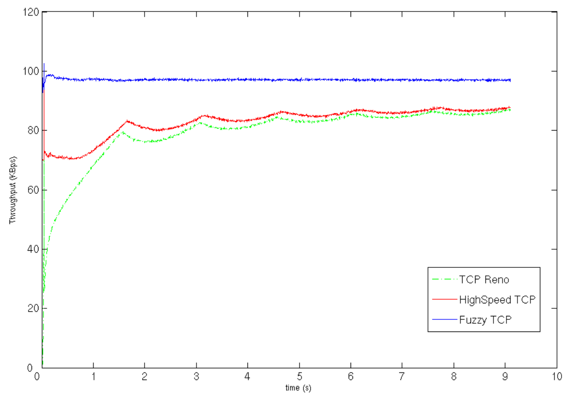

Fig. 6

THROUGHPUT EVOLUTION (SIMULATION)

VI. SIMULATION RESULTS

Simulations of fuzzy rate controllers have been performed by means of the ns-2 [19], a de facto standard within the Internet research community. A performance evaluation study was conducted on traditional and fuzzy rate controllers so as to compare both approaches. We show results for a comparison of TCP Reno [9] (the most extended version of TCP in the current Internet), HighSpeed TCP [12] and the fuzzy rate controller described in this work.

Figure 5 shows a comparison of the congestion window evolution for the three rate controllers being compared, while figure 6 compares throughput of the four TCP variants under the same conditions. A common network scenario was considered for this simulation case. TCP flows are stablished for 10 seconds along a 10 hops long path with $100\,Mbps$ bottleneck bandwidth at the edges and $400\,ms$ average round-trip time. Random losses are simulated in one router in the path. Competing cross traffic with variable sources, path length and round-trip time, consisting of 10% UDP traffic and 90% TCP Reno traffic is used.

As overall conclusions we can draw that the fuzzy extended version of TCP rate control shows higher robustness to loss events. This fact leads to higher final throughput (improved by approximately 12% and 11% as compared to TCP Reno and HighSpeed TCP, respectively). Additional simulations performed on network scenarios under high congestion conditions confirm that the fuzzy rate controller still provides proper and quick reactions to congestion. Common fairness principles considered in the design of Internet end-to-end congestion control schemes [2] are satisfied as well.

VII. EXPERIMENTAL IMPLEMENTATION RESULTS

In order to make the results of our work available for the Internet research community as a research tool, we have developed an experimental user space (instead of kernel space) TCP implementation with fuzzy extensions. A user space implementation allows maximum experimentation flexibility for further refinement of fuzzy rule bases and identification of new rules.

The tool is based on the “Almost TCP over UDP” (atou) [20] utility, developed as part of the web100 project of the High-Performance Networks research program of the U.S. Department of Energy. We introduced changes to atou in order to incorporate the fuzzy inference scheme presented in previous sections. These systems were generated as C code using the Xfuzzy tools. The modified atou tool generates detailed event logs and packet traces with a configurable degree of verbosity.

Modifications for enabling and disabling TCP rate control options and algorithms were also introduced. This way, a flexible framework for experimenting with highly modified implementations of TCP rate control at the application level is available for testing between any two Internet hosts. Among the options that can be set in a text configuration file, the modified version of atou supports a number of TCP rate control variants.

We show a comparison of the three variants of TCP rate controllers that were compared through simulation. Figures 7 and 8 compare the evolution of $cwnd$ and throughput. The network scenario considered is the following: two hosts transfer a file for 10 seconds; the TCP-like connection between the two hosts is stablished along a 5 hops long path. The path bottleneck bandwidth is $100\,Mbps$ and the

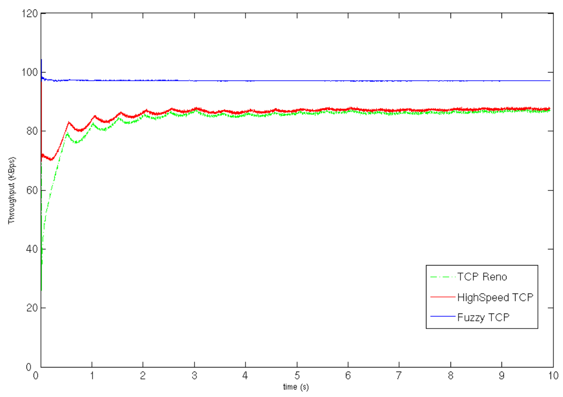

Fig. 8

THROUGHPUT EVOLUTION (IMPLEMENTATION)

average round-trip time is $110\,ms$ approximately. Results from experimental implementation confirm simulation results as for improvements in robustness and throughput. Overall throughput improvement is approximately 15% and 13% with respect to TCP Reno and HighSpeed TCP, respectively.

VIII. CONCLUSIONS AND FUTURE WORK

We have shown that the rate control mechanisms of TCP (the prevalent transport protocol in the Internet), which is currently a major topic of research, can be reinterpreted and extended partially and as a whole in terms of fuzzy logic. The fuzzy model described provides a rule based perspective of current evolving rate control schemes, being the first reported result in the application of fuzzy systems to intelligent network state inference at end nodes.

Both simulation and implementation results show that the proposed fuzzy extension to TCP rate control can improve performance with regards to a number of criteria, namely, faster convergence to achievable transference rate, higher throughput and reduced oscillations around the stabilized rate for long transfers. The proposed fuzzy system also eases finding comprimise solutions for different user requirements.

We note however that there is still a lot of work to do as extensions to the system described in this paper. In particular regarding the identification of new rules, the exploration of the whole set of possible rules and the addition of new inputs. As short term future work we consider the following subjects:

- Application of adjustment and learning techniques to gain further insight on the system dynamics.
- Experimentation with a number of extensions that are current topics of research, such as the initial $cwnd$ value, fuzzy extensions to RTT and retransmission timeout computation[2], and extensions for intelligent loss-congestion differentiation [6], [7]. This work paves the way and provides tools for experimentation with a number of these mechanisms.

[2]Current standard retransmission timer computation employs interpolation techniques to smooth variations in measurements through a simple low-pass filter and take into account RTT variance

ACKNOWLEDGMENT

This work has been supported in part by projects TEC2005-04359/MIC from the Spanish Ministry of Education and Science and TIC2006-635 from the Andalusian regional Government.

REFERENCES

[1] S. Floyd and E. Kohler, “Internet Research Needs Better Models,” in *ACM SIGCOMM First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, USA, Oct. 2002.

[2] J. Wang, D. X. Wei, and S. H. Low, “Modelling and Stability of FAST TCP,” in *24th IEEE INFOCOM*, Miami, FL, USA, Mar. 2005, pp. 938–948.

[3] H. C. Cho, M. S. Fadali, and H. Lee, “Dynamic Queue Scheduling Using Fuzzy Systems for Internet Routers,” in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2005)*, Reno, USA, May 2005, pp. 471–476.

[4] B.-S. Chen, Y.-S. Yang, B.-K. Lee, and T.-H. lee, “Fuzzy Adaptive Predictive Flow Control of ATM Network Traffic,” *IEEE Transactions on Fuzzy Systems*, vol. 11, no. 4, pp. 568–581, Aug. 2003.

[5] P. Carbonell, J. Zhong-Ping, and S. Panwar, “Fuzzy TCP: A Preliminary Study,” in *15th IFAC World Congress*, Barcelona, Spain, July 2002, pp. 21–26.

[6] Ruy de Oliveira and Torsten Braun, “A Delay-based Approach Using Fuzzy Logic to Improve TCP Error Detection in Ad Hoc Networks,” in *IEEE Wireless Communications and Networking Conference*, Atlanta, USA, Mar. 2004.

[7] L. Chang and I. Marsic, “Fuzzy Reasoning for Wireless Awareness,” *International Journal of Wireless Information Networks*, vol. 8 (1), pp. 15–26, Jan. 2001.

[8] V. Jacobson and M. J. Karels, “Congestion Avoidance and Control,” *ACM Computer Communication Review SIGCOMM ’88 Symposium: Communications Architectures and Protocols*, vol. 18, no. 4, pp. 314–329, Aug. 1988.

[9] M. Allman, V. Paxson, and W. Stevens, “TCP Congestion Control,” Internet Engineering Task Force, Network Working Group, RFC 2581, 1999, status: Proposed Standard.

[10] Mark Handley and Sally Floyd and Jitendra Padhye and Jörg Widmer, “TCP Friendly Rate Control (TFRC): Protocol Specification,” Internet Engineering Task Force, Network Working Group, RFC 3448, Jan. 2003, status: Proposed Standard.

[11] R. R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. J. Schwarzbauer, *et al.*, “Stream Control Transmission Protocol,” Internet Engineering Task Force, Network Working Group, RFC 2960, Oct. 2000.

[12] S. Floyd, “HighSpeed TCP for Large Congestion Windows,” Internet Engineering Task Force, Network Working Group, RFC 3649, Dec. 2003, category: Experimental.

[13] D. Leith and R. Shorten, “H-TCP: TCP for High-Speed and Long-Distance Networks,” in *Second International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet)*, Feb. 2004.

[14] T. Kelly, “Scalable TCP: Improving Performance in Highspeed Wide Area Networks,” *ACM Communication Review*, pp. 83–91, Apr. 2003.

[15] D. Rolls, G. Michailidis, and F. Hernndez-Campos, “Queueing Analysis of Network Traffic: Methodology and Visualization Tools,” *Computer Networks*, vol. 48 (3), pp. 447–473, June 2005.

[16] A. Cabrera, S. Sánchez-Solano, P. Brox, A. Barriga, and R. Senhadji, “Hardware/Software Codesign of Configurable Fuzzy Control Systems,” *Applied Soft Computing*, vol. 4, no. 3, pp. 271–285, Dec. 2004.

[17] F. Moreno-Velo, I. Baturone, S. Sánchez-Solano, and A. Barriga, “Rapid Design of Fuzzy Systems With Xfuzzy,” in *FUZZ-IEEE’03. The 12th IEEE International Conference on Fuzzy Systems*, May 2003, pp. 342–347.

[18] F. Moreno-Velo, S. Sánchez-Solano, A. Barriga, I. Baturone, and D. López, “XFL3: a New Fuzzy System Specification Language,” in *5th WSEAS/IEEE Multiconference on Circuits, Systems, Communications and Computers (CSCC’01)*, Rethymon, July 2001, pp. 361–366.

[19] Information Sciences Institute. University of Southern California, Viterbi School of Engineering, “The Network Simulator – ns-2,” http://www.isi.edu/nsnam/ns/, Feb. 2006.

[20] T. Dunigan, F. Fowler, *et al.*, “Almost TCP over UDP (atou),” Oak Ridge National Laboratory, U.S Department of Energy, Feb. 2006, http://www.csm.ornl.gov/ dunigan/net100/atou.html.