International Journal of Information Technology and Knowledge Management
January-June 2011, Volume 4, No. 1, pp. 151-156

CONGESTION CONTROL IN DIFFI-SERV USING FUZZY AND NEURAL METHODOLOGY

Aastha[1], Tarun Aggarwal[2] & Sachin Garg[3]

The paper covers a brief introduction of the problem of congestion and its control using the rule base of the fuzzy module refined by neural module. A number of new applications being developed for the Internet require stringent bounds on parameters such as the end-to-end delays and inter-packet jitter. In order to provide these bounds led to the development of alternate mechanisms based on policies that could be deployed on the existing network infrastructure with minimal changes, and still provide a suitable Quality of Service. Differentiated Services (Diff-Serv) is one such technology. DS can be implemented with relatively small disturbance to the existing infrastructure. For this the congestion control algorithm called Random Early Detection is discussed with fuzzy-neuro system as the technique called Random Early Detection technique used in differentiated services network by applying Neuro-Fuzzy approach is slightly modified. First, Introduction to fuzzy inference module by applying various fuzzy based control strategies is there and then refined with its rule-base with the help of neural network to incorporate network dynamic conditions which keeps on changing with time. The experience with NS2 network simulator is also discussed.
Keywords: Congestion Control, Differentiated Services, Neuro-Fuzzy Systems

1. INTRODUCTION

1.1. Congestion

Congestion control should not be confused with flow control, which prevents the sender from overwhelming the receiver. Congestion control is concerned with allocating the resources in a network such that the network can operate at an acceptable performance level when the demand exceeds or is near the capacity of the network resources.

In simple terms, "Congestion occurs when routers receives packets faster than they can forward."

Although resource allocation is necessary even at low load, the problem becomes more important as the load increases because the issues of fairness and low overhead become increasingly important.

1.2. DiffServ

Differentiated Services or DiffServ is a computer networking architecture that specifies a simple, scalable and coarse-grained mechanism for classifying, managing network traffic and providing Quality of Service (QoS) guarantees on modern IP networks.

[1]Department of Computer Science & Engineering, TIT&S, Bhiwani, India

[2]Tata Consultancy Services, Gurgaon, India

[3]Global Systems LLC, Irwing, TEXAS, US

Email: [1]astha.raghu@gmail.com, [2]tarun.002@gmail.com, [3]mail2sac_garg@yahoo.com

DiffServ operates on the principle of traffic classification, where each data packet is placed into a limited number of traffic classes. Each router on the network is configured to differentiate traffic based on its class. Each traffic class can be managed differently, ensuring preferential treatment for higher-priority traffic on the network.

DiffServ can, for example, be used to provide low-latency, guaranteed service (GS) to critical network traffic such as voice or video while providing simple best-effort traffic guarantees to non-critical services such as web traffic or file transfers.

Basic Goals of Diffi-Serv:

- It allows different levels of services on a common network infrastructure.
- It provides a better QoS.

2. SIMULATION DETAILS

2.1. Introduction

In the work proposed here is to control the congestion using fuzzy rule set and refining these rule set with the help of neural network module, for this, the previously used congestion control technique called Random Early Detection for Diffi-Serv is used. The RED implementation for Diffi-Serv defines different thresholds for each class. RED simply sets some minimum and maximum dropping thresholds in the router queues. If the buffer queue size exceeds the minimum threshold, RED starts randomly dropping packets based on a probability depending on the average queue

length. If the buffer queue size exceeds the maximum threshold then every packet is dropped.

2.2. The Implementation

The implementation goes in the two modules. First is Fuzzy Inference module and other is Neural network Adaptive Module which is wholly be called as Neuro-fuzzy system.

Fuzzy logic system deals with explicit knowledge that is average queue length in this case, and neural networks, which deal with implicit knowledge that are fuzzily calculated packet drop probability and error signal in this case, which can be acquired by learning.

Whenever a packet arrives to get enque in the buffer then average queue length is calculated by the RED algorithm. If average queue length is lower than the minimum threshold then packet is simply enqued. But if when buffer average queue size exceeds the minimum threshold, at that time concept of fuzzy logic to calculate the packet drop probability is introduced.

First of all Fuzzy logic fuzzifies the average queue length and calculates packet drop probability through the inference engine which works on rules defined in rule-base. Finally the calculated fuzzified packet drop probability is defuzzified by the defuzzifier of fuzzy logic Module. This defuzzified packet drop probability is used as parameter to decide whether packet will be enqued or early-dropped.

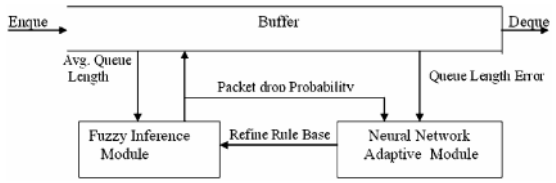


Fig. 1: The Basic Approach used for the Implementation

2.3. Implementation of Fuzzy Inference Module

Fuzzy logic module starts working whenever a packet comes and the calculated average queue length is greater than minimum threshold and less than maximum threshold. It fuzzifies the average queue length and calculates packet drop probability through inference engine.

Fuzzy logic is especially suitable for cases where control decisions must be made without exact information about the controlled environment. Fuzzy inference systems consist of fuzzy variables, a rule base and a reasoning mechanism. This fuzzy inference module modifies its rule-base according to the input provided by neural network. Fuzzy Inference Module comprises of four sub-modules.

- Fuzzifier;
- Inference Engine;
- Rule-Base;
- Defuzzifier.

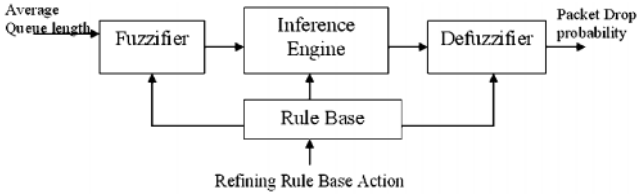


Fig 2: The Fuzzy Inference Module

2.4. Fuzzifier

Fuzzification is necessary to convert the input crisp data into a suitable set of linguistic value that is needed in inference engine. Linguistic values separate numeric value ranges of the variables to sub-ranges. The input provided to the fuzzifier is the average queue length calculated.

The term set of input linguistic variable can be defined as:

fuzzy_len(ql)= {low, medium, high, very high}

The fuzzy value represented by fuzzy_len[ql] can be written as (low < medium) where ("<" fuzzily means small, not absolutely small). So the queue length (low) is fuzzily minimal and (very high) means fuzzily maximal. The centre value of fuzzy_len is denoted by qc[i]. Thus qc[i] < qc[i+1] (where "<"fuzzily means small, not absolutely small). The larger value of i means the heavier congestion in the link, and then much larger Pkt_drop_prob (pdp) is required.

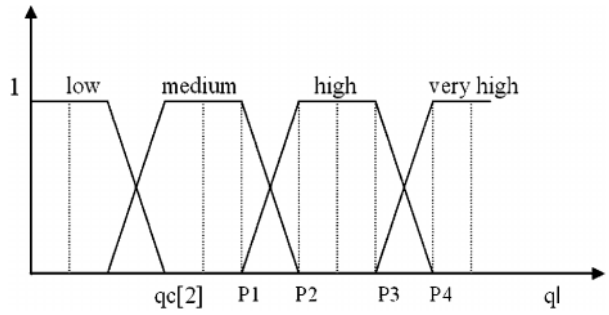


Fig. 3: Trapezoidal Function

To fuzzify the input variable, have been used the trapezium function as membership function. Trapezium function takes five parameters as input and output one linguistic value which is fuzzified form of average queue length.

2.5. Rule-Base

A rule base of fuzzy controllers defines control actions of controllers in the defined states of controlled processes. The rule base consists of a set of IF-THEN type of rules which

describe the controlled states of the processes and suitable control actions for the states using the linguistic values of input and output variables. As we know that the fuzzy_len(ql) signify the current congestion status at some degree. The larger the value of i in fuzzy_len means heavier congestion in the link, and then network require much larger packet drop probability pdp[i]. So the rules are defined as:

- If average queue length is low then packet drop probability is small.
- If average queue length is medium then packet drop probability is medium.
- If average queue length is high then packet drop probability is high.
- If average queue length is very high then packet drop probability is very high.

2.6. Inference Engine

It is used to infer the value of the parameter according to the applicable rule specified in the rule-base. The value can be calculated on the fuzzified average queue length by the rule-base. First the input variable qParam_[prec].edv_.v_ave is first mapped to its membership function and then the final output of the FIM, Packet drop probability is calculated as:

Pkt_drop_prob =

$$\frac{i = 1 \ldots 4 \sum \text{fuzzy\_len\_cal}[i] * \text{prob}[i]}{i = 1 \ldots 4 \sum \text{fuzzy\_len\_cal}[i]}$$

2.7. Defuzzification

An output variable in fuzzy logic controllers describes possible control actions. As fuzzy variables, consists of one to several linguistic values which are commonly understandable names or symbols. So in this particular step we defined output term set of the variable Pkt_drop_prob(pdp). The term set of packet drop probability, as follows.

Pkt_drop_prob (pdp) = {small, medium, high, very high}

The fuzzy value represented by Pkt_drop_prob(pdp) can be written as (very small < medium) where ("<" fuzzily means small, not absolutely small). So the packet drop probability (small) is fuzzily minimal and (very high) means fuzzily maximal. The centre value of Pkt_drop_prob(pdp) is denoted by pc[i]. Thus pc[i] < pc[i+1] (where "<"fuzzily means small, not absolutely small). We map the Pkt_drop_prob[i] to the crisp packet drop probability. This defuzzified packet drop probability is used to determine whether to drop packet or enque it.

2.8. Implementation of Neural Network Adaptive Module

To handle changing network conditions, the need is to update the network parameters time to time. For that, apply second part of the algorithm which is refinement of rule-base through neural network to help inference engine to take action according the current network condition. It aims to minimize the error signal, e between actual queue size and stable queue size. The back-propagation learning algorithm has been used to update the network weights and bias.

2.9. The Model

Here, have been used a dynamic recurrent neural network with feedback connection. The model uses three-layer perceptron neural network. The inputs of this neural network are the error signal, e and the probability, p as a feedback signal from its output. Here, have been treated the input signal as one-dimensional array u = [e, prob_output]. The initial values of the weights in the first and second layers, and the bias, were uniformly distributed in [–1, 1].

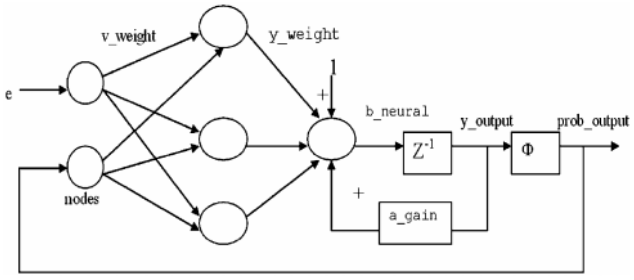


Fig. 4: The Neural Network Adaptive Module

The weights of first layer are in two-dimensional matrix form and are represented by v_weight. The first column and the second column of the first layer are related to the error signal and the feedback probability, respectively. The weight vector in the second layer is a one-dimensional form represented by y_weight. We have taken b_neural as a bias connected with unit input. The dynamic behavior of the network is given by:

$$y\_output_{i+1} = a\_gain * y\_output_{i} + y\_weight^{t} * (v\_weight * u) + b\_neural$$

where a_gain is the feedback gain, i denotes discrete time.

Finally, the network prob_output is obtained from the activation function is given by:

$$prob\_output = 1/(1 + \exp(-(a\_scaling * y\_output)))$$

where a_scaling is a positive constant scaling factor. This particular output is used as input for the rules modification in rule-base. This prob_ouput is also sent back to the network as a feedback.

2.10. The Training Procedure

Back-propagation learning algorithm is used to derive the rules for updating the network weights and bias. The neural network was trained to determine the optimal weights and bias after iterative network training with randomly chosen initial weights and bias, the optimal weights that minimize the error signal.

The back-propagation algorithm works on the principle of computing activations and signals of input, hidden and output neurons in the sequence. Then it computes the error over the output neurons by comparing the generated outputs with the desired outputs. This computed error is used to change in the hidden to output layer weights, and change in input to hidden layer weights. The bias weights which are connected to network are also updated.

2.11. The Weight Updates

For the hidden to output layer weights:

$y\_weight^{k+1} = y\_weight^{k} + \Delta y\_weight$

$\Delta y\_weight = n\_learning\_rate * delta * (\Sigma_{i=12,3,\ j=1,2}\ v\_weight_{ij} * u[j])$

For the input to hidden layer weights

$v\_weight^{k+1} = v\_weight^{k} + \text{“}v\_weight$

$\Delta v\_weight = n\_learning\_rate * delta * (y\_weight_{i} * u[j])$

where n_learning_rate is the learning rate and delta is

$delta = q\_error * (q\_error \setminus (prob\_output_{(k)} - prob\_output_{(k-1)})) * (a\_scaling *$

$(exp(–(a\_scaling*y\_output)))/(1+ (exp(–(a\_scaling * y\_output))))^2)$

2.12. The Bias Update

$b\_neural^{k+1} = b\_neural^{k} + \Delta b\_neural$

$\Delta b\_neural = n\_learning\_rate * delta$

2.13. Refinement of the Rule-base

pc[3] = minimum(1, temp_pdr)

where temp_pdr = prob_output * ((qc[3]-qc[0])/(q_stable-qc[0]))

If qc[i] <= q_stable

Then pc[i] = (prob_output * ((qc[j]-qc[0]) / (q_stable-qc[0])

If qc[j] > q_stable

Then pc[i] = ((pc[3]* (qc[j] - q_stable))/2 + ((prob_output * (qc[3]-qc[j])) / (qc[3]-q_stable)))

Where i = 0…3

2.14. Refining Rule-base when Queue Average is Greater than Maximum Threshold

When the average queue length is greater than the maximum threshold then updated the rules in the rule-base of fuzzy logic module. But difference in this rules updates is that here I modify the linguistic queue length parameter.

If queue_average >= max_threshold

Then q_stable = (min_threshold +max_threshold)/2

q [0] = min_threshold

q [3] = max_theshold

and other q[i] are updated in the arithmetic progression.

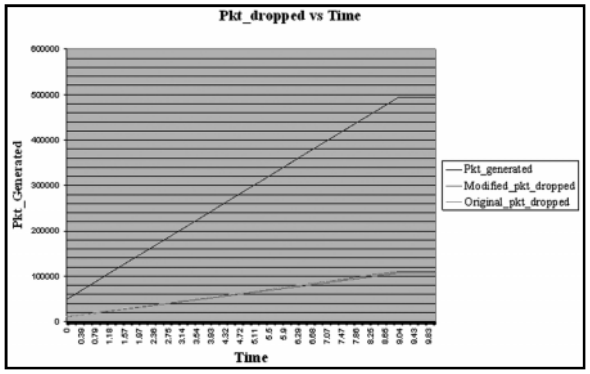
3. RESULT AND DISCUSSION

Analysis of the performance of both the algorithms; original RED and the Modified RED is here with some cases.

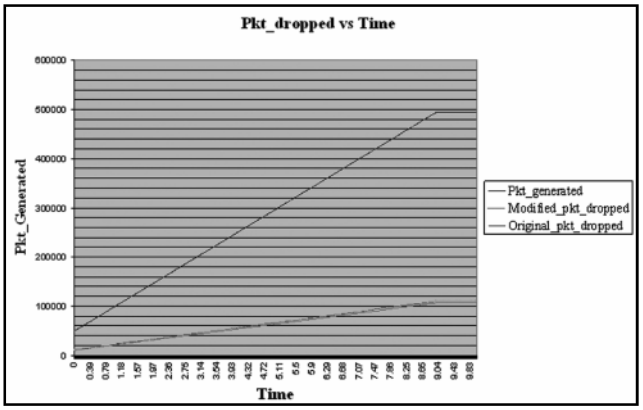
Case1: Two CBR EF traffic flow sending packets from S3 to D2 and S6 to D3 at 1.5Mbps; And four CBR best-effort traffic Sending from S1 to D1, S2 to D1, S4 to D2, S5 to D3 at 31Mbps. The scheduler used is priority and queue rate is 30Mbps.

Table1
Results Case 1

Algorithm	TotPkts	TxPkts	ldrops	edrops
Original_dsRed	494595	384155	502	109938
Modified_dsRed	494595	387750	79974	26871



After calculations, the percentage of packets dropped in original RED is 22.329% and that of modified RED is 21.602% in this case & The Throughput of original RED = 77.6706% and that of modified RED =78.3975%.



Case 2: The traffic sending rate of two CBR EF is same as case1. But changed the rate of four CBR best-efforts to 25Mbps. The scheduler used is priority again and queue rate is 30Mbps.

Table 2
Results Case 2

CP	TotPkts	TxPkts	Idrops	edrops
Original_dsRed	401129	384542	1230	15357
Modified_dsRed	401129	388464	23	12642

The Throughput of original RED = 95.864% and that of modified RED = 96.843%, the packet drop probability of original RED comes out 4.135% and that of modified RED is 3.157, which is good enough.

So, the results show that performance can be improved and in future, we can have a better version to be implemented on a Diffi-Serv network.

REFERENCES

[1] A. Pitsillides, A. Sekercioglu, C. Chrysostomou, G. Hadjipollas and M. Polycarpou, "Fuzzy Explicit Marking for Congestion Control in differentiated Services Networks", in Proc.8th IEEE Symposium on Computers and Communications, Antalya, Turkey, 1, pp. 312-330, June-3 July 2003.

[2] Raj Jain, "Congestion Control in Computer Networks: Issues and Trends," IEEE Network Magazine, pp. 24-30, May 1990.

[3] A. Pitsillidesa, A. Sekerciogluc, C. Chrysostomoua, L. Rossidesa and M. Polycarpoub, "Congestion Control in Differentiated Services Networks using Fuzzy-RED," [Online]. Available www.sciencedirect.com, 2003.

[4] Hannu Koivisto, Mikko Laurikkala, Teemu Ekola and Timo Lehto, "Performance of Nonlinear Queue Management Algorithm in Best-effort Networks," Tampere University of Technology, Finland, 2005.

[5] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Trans. Networking, 4, pp. 397–413, Aug. 1993.

[6] C.V. Hollot, Don Towsley, Vishal Misra, and Wei-Bo Gong, "A Control Theoretic Analysis of RED," in Proc. IEEE INFOCOM, 2001.

[7] A. Pitsillides, C. Chrysostomou, L. Rossides, and Y. A. Sekercioglu, "Fuzzy Logic Controlled RED: Congestion Control in TCP/IP Differentiated Services Networks," Soft Computing Journal, Springer-Verlag, 8, pp. 79-92, December 2003.

[8] Farhan Shallwani, Jeremy Ethridge, Mandeep Baines and Peter Pieda, "A Network Simulator Differentiated Services Implementation Open IP, Nortel Networks," July 26, 2000.

[9] Manish Mahajan, Ananthanarayanan Ramanathan and Manish Parashar, "Active Resource Management for the Differentiated Services Environment", International Journal of Network Management, pp. 149–165, 2004.

[10] Jun OGAWA, Yuji NOMURA, "A Simple Resource Management Architecture for Differentiated Services".

[11] Gonzalo R. Arce, Kenneth E.Barner, and Liangping Ma, "Median Red Algorithm for Congestion Control," IEEE 2004.

[12] Andreas Pitsillides and Ahmet Sekercioglu, "Fuzzy Logic based Congestion Control," in Proc. COST 257: Impacts of New Services on the Architecture and Network Performance of Broadband Networks, Larnaca, Cyprus, September 1999.

[13] A. Pitsillides, A. Sekercioglu, C. Chrysostomou, G. Hadjipollas and M. Polycarpou, "Fuzzy Logic Congestion Control in TCP/IP Best-Effort Networks", in Proc. Australian Telecommunications Networks and Applications Conference, Melbourne, Australia, 8-10 December 2003.

[14] David Lapsley and Steven Low, "Random Early Marking: An Optimization Approach to Internet Congestion Control".

[15] Dong Lin and Robert Morris, "Dynamics of Random Early Detection," Slightly Revised Version of the Paper Appeared in Proc. of SIGCOMM,1997.

[16] "AFRED: An Adaptive Fuzzy-based Control Algorithm for Active Queue Management", Proceedings of the 28th Annual IEEE International Conference on Local Computer

Networks (LCN'03), 0742-1303/03 $ 17.00 © 2003 IEEE.

[17] "Neuro–Fuzzy Rule Generation: Survey in Soft Computing Framework" Sushmita Mitra and Yoichi Hayashi, IEEE Transactions on Neural Networks, 11, No. 3, May 2000.

[18] "Neural Network Control for TCP Network Congestion", Hyun C. Cho, M. Sami Fadali, Hyunjeong Lee, 2005 American Control Conference, June 8-10, 2005. Portland, OR, USA.