A Neural Networks Approach to the Estimation of the Retransmission Timer (RTT)

George Develekos[1], Othon Michail[2], Christos Douligeris[3]

[1] Cyberstream Ltd., 75 Lydias St., 16121 Athens, Greece
Email: george.develekos@cyberstream.gr
[2] 36 Xatzopoulou St., 34100 Chalkida, Greece
Email: othon@mycosmos.gr
[3] University of Piraeus, Piraeus, Greece
Email: cdoulig@unipi.gr

Abstract. In this paper we have used a Neural Networks approach for the estimation of the Round Trip Time (RTT) in the TCP protocol. We constructed a network operation scenario that approaches the Internet operation. By observing the traffic between different nodes we collected the RTT values during normal network operation. Then we organized these values in such a way so that part of them was used as a training set of the Neural Network (NN) and the rest for the evaluation of the performance of the training of the NN. The next step was the combination of the ns2 source code with the NN source code in order to replace the Jacobson's algorithm with the NN approach in the procedure of the RTT estimation. We operated again the previous simulated network with three different scenarios, alternating between the NN and Jacobson's algorithms. The results show that the Neural Network approach provides a better match than Jacobson's algorithm.

1 Introduction

One of the most pressing problems in the study of networks' operation is the control of the data traffic volume, aiming to prevent congestion avoidance and to increase the overall performance [8]. In this paper we focus on congestion control in the Transport Control Protocol (TCP). The extensive use of this protocol in the Internet has shown that the most often used congestion control algorithm – Jacobson's algorithm [1] – is very efficient and affords good stability. There is, however, plenty of room for improvement. In this paper we propose a new solution to the problem in question. This solution is based on a Neural Networks approach. We propose the replacement of the Round Trip Time (RTT) estimation mechanism with a properly trained Neural Network, aiming for a better estimation of RTT and therefore a more efficient operation of the protocol. Comparison trials were carried out using the ns2 tool that was chosen because of the availability of its source code, which allowed us to incorporate the source code of our Neural Network. We constructed a suitable and representative scenario of network traffic and compared the performance of a conventional network to one that implemented the NN RTT estimation algorithm. Then, based on results of

the simulation with ns2, we have compared the data that characterize the efficiency of the protocol, such as the total number of successfully transmitted packets, the total number of retransmissions, the total number of bad (not justified) retransmissions and the final throughputs for all network flows. The results show that the Neural Network approach provides higher performance than JACobson's algorithm. Although bad retransmissions in the Neural Network approach are slightly higher than JACobson's algorithm, this cost is insignificant compared to the profit gained from the total pure throughput of the Neural Network approach. Final throughputs results show that this cost does not adversely affect the overall network traffic.

1.1 Congestion Control on TCP/IP

During the transmission of data packets through an unreliable network (such as the Internet) two undesirable facts can take place. Either because of excessive 'noise' the packets can be corrupted or because of various factors (for instance overflowing of routers due to congestion) the packet cannot reach its destination. In both cases, we have to deal with transmission errors that can be handled by a protocol intelligent enough to protect the network operation from errors. In fact, in TCP/IP, this work is attributed to TCP, seeing that IP can not give any guarantee concerning the arrival of the packet to its destination without loss. For the first case, that is to say the corruption of the data packet, TCP uses the check sum. But when a packet does not reach its destination or arrives with great delay then the problem is due to the regulation of the retransmission time and generally to the avoidance of congestion an the retransmission of the packets without great delay.

When a packet does not arrive to its destination, neither the source nor the destination knows it. The TCP source entity must wait until it receives an ACK from the destination, confirming that the packet has arrived successfully. If there is no a response from the destination, then the source must retransmit the packet. A basic topic is consequently the retransmission timer. If this time is very long then the response of TCP in lost packets will be too late. If this time is too short then there will be many useless retransmissions, increasing the traffic in the Network and in the same time the danger of congestion. This time must be a little longer than the round trip delay, that is to say the time that is necessary for the packet to reach its destination and for a confirmation of the receiving end to arrive at the source [5]. The use of a fixed value for this time does not represent real network conditions, which usually result in large variabilities due to changes in traffic conditions or changes in the paths of messages. A better approach is the observation of the delay of the most recently sent packets. Thus we are able to estimate the RTT for a new packet in a more accurate and realistic way, since we base our estimate on the conditions that currently prevail in the network.

In RFC 793 [11] it is proposed that the Round Trip Time is estimated on the basis of the previous round trip delay times (RTO). The algorithm calculates an exponentially weighted average value of the previous RTT, emphasizing the recent values that probably reflect the future behavior of the network, as follows:

$$SRTT\ (k+1) = \alpha * SRTT\ (k) + (1-\alpha) * RTT\ (k+1) \quad \textbf{(1)}$$

where $SRTT\ (k)$ is the normalized estimation of the round-trip time and $0 < \alpha < 1$. The above equation can be written in more detail as follows:

$$SRTT\ (k+1) = (1-\alpha) * RTT\ (k+1) + \alpha(1-\alpha) * RTT\ (k) + \ldots + \alpha^{k}(1-\alpha) * RTT\ (1) + \alpha^{k+1}(1-a) * RTT\ (0) \quad (2)$$

As we can conclude from this equation, the older the observation the less it contributes to the estimate. Using the above equation we can calculate the next RTT, but the value of the timer must be higher than the evaluated RTT. According to RFC 793 the retransmission time out (RTO) is calculated as follows:

$$RTO\ (k+1) = \min(UBOUND, \max(LBOUND, \beta * SRTT\ (k+1))) \quad (3)$$

The perfect estimation of the RTT and RTO will not provoke congestion on the network due to useless retransmissions of packets. At the same time a good estimation will give the opportunity to the protocol to complete quickly any data retransmission, that is to say to increase the response of the protocol but not at the expense at the network's traffic.

The above approach deals with many problems in case of sudden explosion of the network traffic due to imponderable factors. Jacobson [1] as well as Karn [2] have tried quite successfully to deal with the above cases, using appropriate algorithms.

According to the technique described in [1], the entity TCP is adapted to the changing of the round trip time. Moreover, in order to deal with the potentially large variability in the delays in the TCP specifications, the RTT estimator is multiplied by a constant β:

$$RTO\ (k+1) = \beta * SRTT\ (k+1) \quad (4)$$

where usually $\beta = 2$. Another means of estimation of the average range is through

$$MDEV\ (x) = E\left[\left|x - E(x)\right|\right] \quad (5)$$

As we have done in the case of the RTT estimation a simple mean can be used in order to estimate MDEV:

$$AERR\ (k+1) = RTT\ (k+1) - ARTT\ (k) \quad (6)$$

$$ADEV\ (k+1) = \frac{1}{(k+1)\sum_{i=1}^{k+1}\left|AERR\ (i)\right|} = \frac{k}{(k+1)\,ADEV\ (k)} + \frac{1}{(k+1)\left|AERR\ (k+1)\right|} \quad (7)$$

where $ARTT\ (k)$ is a simple mean and $AERR\ (k)$ is the indicated average divergence calculated during time k. Experience has shown that the algorithm can considerably improve the efficiency of TCP [8].

The basic philosophy of above algorithm is the estimation of the existing traffic on the network through the round trip delay times of the previous packets. But there is a main disadvantage in these methods. They try with a linear model, to anticipate a group of data that are not linearly connected between them and so they can not respond efficiently to unexpected situations. Essentially, the problem of anticipating RTT and RTO times is a problem of anticipating chronological ranges. In such problems Neural Networks demonstrate a remarkable efficiency [7].

Neural Networks belongs to the family of generalized non-linear modeling. Fuzzy logic systems have also been applied in similar situations [9]. On a theoretical basis NNs can approximate every non-linear function with an arbitrary precision [12]. It has been noted that NNs can have a great influence on the modeling and the anticipation of no-linear chronological ranges with or without noise. NNs are particularly flexible according to linear or non-linear prediction methods for chronological ranges because they are conducted by the data themselves and demand few a priori conditions for the models the reach. Instead of trying to discover the relation that condition the data, they leave the data "talk" about themselves and in this way they are better adapted to them it seems then that we have in air hands an alterative tool of estimation and prediction old RTT values that is proved more efficient than the solutions that have been applied until now.

2 Estimation of RTO with NN Approach Method

In the previous section we raised the point of accurate estimation of RTO and we presented the methods that have been used for its efficient solution. We have also noted the remarkable efficiency of the Neural Networks in the prediction of chronological ranges. In this section, we will describe in great detail the methodology that we applied in order to estimate the RTO using a Neural Networks approximation.

2.1 General Methodology Plan

First we gather a representative sample of RTT values. This is done by simulating the operations of a network that works according to the protocol TCP/IP. The tool that we use for the simulation is the ns2 (ver. 2.1b8a-win) [10]. Then with a part of this sample we train the NN and with its remaining part we check the training efficiency. According to the results of the training control, we choose a NN and we incorporate it to ns2. The NN will calculate RTT values for a certain flow in the network. We also modify the RTO calculation for this flow. Then we run successive simulations of the same network from which we collected RTT values for different times. For each running we used both the NN approach and then Jacobson's algorithm (JAC). We calculated the total number of packets that were delivered to their destination, the total number of retransmission packets, the number of the correct retransmissions (that is to say the packet that indeed has been lost) and the number of retransmissions that are incorrect (that is to say, the packets that have reached their destination, but the source has not be updated in time) for the flow that we are watching. We also calculated the throughputs of all the network flows. From the comparison of these data we were able to draw valid conclusions about the merits and demerits of our approach.

2.2 NN training Data Gathering

While it is obviously hard – if not impossible – to simulate the entire Internet, it is usually sufficient to simulate a smaller network model that has most of the Internet attributes on a small scale.

2.2.1 Network Simulator ns2

The simulations were run using the Network simulator ns2 (ver. 2.1b8a-win). This tool has been chosen because it is tried and tested in TCP research and offers many possibilities of discovering protocol operation parameters. This tool has been developed in C++ and Objective Tcl (otcl). The fact that the source code is available, gives us the opportunity to experiment with the details concerning the structure of TCP protocol. This fact will also help us adapt the Neural Network that we have constructed (implemented in C++ as well) to the TCP operation, replacing JAC as the RTT calculation formula for a single flow that we focus on.

2.2.2 Network Simulation – Topology, Parameters

The topology that we have chosen is shown in figure 1. This topology has been used in several works on TCP/IP, such as the ones given in [10] and [4].

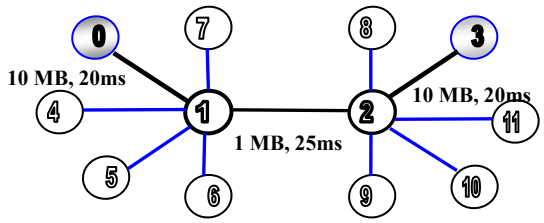

Fig. 1. The network topology

As we notice, the topology is composed of 12 nodes. Nodes 1, 2 are the routers. All the nodes on the left communicate with the nodes on the right and vice versa. All the connections are duplex (as TCP demands). There are several 'strategies' of dealing with tails of packets such as FIFO, RED and Drop Tail. The strategy that we use here is the Drop tail, that is to say the rejection of the surplus packets if the total number of the tail packets surpasses a certain limit. The limit in our network is 30 packets. With this number we certainly have rejections of some packets when there is congestion as it is exactly the case in the Internet. The capacity of each connection between the routers is 10MB and between the routers is 1MB. FTP, TELNET flows and HTTP applications will appear to the network in order to closely simulate a real-life network [3]. More specifically there are 17 FTP flows and 22 TELNET flows. The sources, the destinations and the ID numbers of its flow are shown in Table 1.

As we notice, each flow has its own flow id and flows start and finish at random times, that can be manually given through the ns2 simulator. The HTTP traffic is also randomly fixed by ns2 between nodes-clients number 7, 0, 4, 5, 6 and nodes-servers number 8, 3, 10, 11, and 9. The only flow that we control when it starts is the FTP flow with id=200 between nodes 0 and 3. We monitor carefully this flow, because from its operation we take RTT values that have been fixed by TCP/IP thanks to Jacobson's algorithm. We will set up these values for the NN training process. For that reason we adopt a ns2 Agent that gives us values for a range of parameters concerning RTT. In this flow we will adapt the NN, so that RTT and RTO values will be calculated by it. The results correspond to the operation of the simulated network for 12000

sec. The RTT values gathered from the filtering are suitably organized. Through them we train the NN and later we check the efficiency of its training.

Flow id	FTP 37	FTP 38	FTP 39	FTP40	FTP 41	FTP 42	FTP 43
Nodes	4→8	4→8	4→8	4→8	7→9	7→9	7→9
Flow id	TELNET 53		TELNET 54		FTP 44	FTP 45	FTP 46
Nodes	8→4		8→4		7→9	11→6	11→6
Flow id	TELNET 55		TELNET 56		TELNET 57		FTP 47
Nodes	8→4		8→4		9→7		11→6
Flow id	TELNET 58		TELNET 59		TELNET 60		FTP 48
Nodes	9→7		9→7		9→7		11→6
Flow id	TELNET 61		TELNET 62		TELNET 63		FTP 49
Nodes	10→5		10→5		10→5		10→5
Flow id	TELNET 64		TELNET 65		TELNET 66		FTP 50
Nodes	10→5		11→6		11→6		10→5
Flow id	TELNET 67		TELNET 68		TELNET 69		FTP 51
Nodes	11→6		11->6		4→8		10→5
Flow id	TELNET 70		TELNET 71		TELNET 72		FTP 52
Nodes	4→8		4→8		4→8		10→5
Flow id	FTP 200	**Nodes**	0→3				

Table 1. The sources, destinations and ID numbers of network's flows.

2.2.3 Organizing RTT Values

The organizing of RTT values is done in such a way that they correspond to the NN topology and the way a NN functions. Using these values we train the NN and then we check the success of this training.

Before we proceed to the data organization, it is necessary to present the structure of the neural system that has been chosen. The NN that we will train has 5 inputs nodes, 10 hidden nodes, and 1 output node given that we are interested in the prediction of one value in each round of the NN operation. In the figure 6 we can see the schematic representation of our NN:

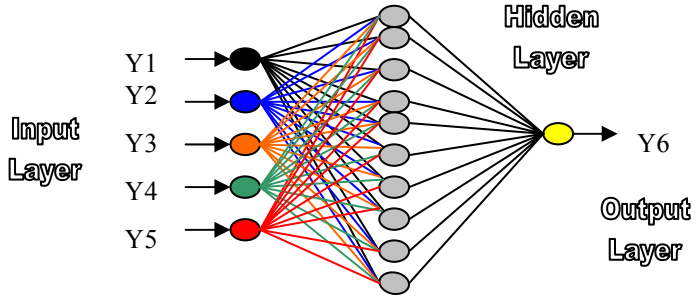

Fig. 2. The structure of Neural Network

Therefore, the data must be organized in 6-value sets, where the first five values will be the input data, and the 6th value will be used in order to calculate the error according to the value produced by NN. So the 6th value will be the target value. The NN is

obliged to predict the 6th value from the five input values that it receives. There is a constant flow of values into the NN. At each instant a new value enters and one of the past values disappears. The 6-value sets, when the training data and the control of training will be organized, must be dependent. If for instance we have the following RTT values: 1 2 3 4 5 6 7 8 they will be organized as follows:

Input Values					Target Value
1	2	3	4	5	6
2	3	4	5	6	7
3	4	5	6	7	8

Table 2. How we organized RTT values.

Initially we will have the first five values like input values and the 6th value like target-value for the NN. Then in the next training prediction step, this value passes to the 5th input value, with a sliding of the input data to the left. The number 7 is the new target value.

It is also very important to determine the total number of RTT values, gathered from the simulation. This number will form a representative section of the simulated network operation. It is obvious that the more representative the section is, on the basis of which a NN will be trained, the better it can respond to the total network function, making accurate predictions even in unexpected situations that may occur. The representative section of RTT values is the 'yeast' for the successful training of NN. Therefore, we choose a number in the range of 70000 6-value sets. The choice is not arbitrary. It is done

1. according to the calculating power of the computer, and
2. according to the total width of RTT values, resulted from the simulation of the previous network in ns2

2.3 NN training – Parameters, results

After gathering and organizing the training data, we are ready to begin the construction – training of the NN. In the Back – Propagation algorithm there is a continuous repetition of the algorithm until the satisfactory convergence according to some criteria. If these criteria are very strict then we may end up with a NN that will be totally dependent on the training values, and of course this fact will reduce its adaptability to the various data it will subsequently be called to handle in full operation. If the criteria are met too easily, then it is certain that the NN will not have time to 'learn' correctly the relation that govern the data.
In our case we use four criteria:

1. *GRADTH=7.2300000E-006* Termination condition—Convergence limit
2. *ANATH=0.0* Termination condition – Analogical error
3. *MAXTH=0.0* Termination condition —Maximum error
4. *ITERTH=60000* - Number of internal repetitions

If one of these internal repetitions is met, the training stops. The above criteria have been chosen after many trials and fine tuning, following standard literature procedures [6].

At the end of the training process we have 71 values that correspond to the weights and the biases of the Neural Network. These values (11 biases and 60 weights) have been formed during the neural network's training and form its core. From that point we move on to the NN 'quality' training control.

For the training progress, we have used the rest of the values section, resulted from the simulation of our network in 12000 sec. It is about 25200 6-value sets that have been organized in the way that we described above. Specifically we 'give' now the first five values to the NN and we compare the Net Value with the target value. We use the following control criteria:

- Regulated error percentage according to the model:

$$error = \frac{|Net - Target|}{Target} * 100\% \quad (8)$$

- The average of the errors that have been calculated according to the model 7:

$$mean\_error = \frac{\sum_{n=1}^{nVPATTERN} error_n}{nVPATTERN} \quad (9)$$

- The minimum and maximum error that have been calculated according to the model 7:

$$\min\_error = \min(error_1, error_2, \ldots, error_{nVPATTERN}) \quad (10)$$

$$\max\_error = \max(error_1, error_2, \ldots, error_{nVPATTERN}) \quad (11)$$

where *nVPATTERN* is the total number of the control training data set.

- The error value, that is higher than 99% of all the others error values that have been calculated according to the model 7.

For our NN the training control has given us the following results:

```
   ******************************************************
   * Statistics Notes *
   ******************************************************
   Number of Layers --> 2
   Number of Neurons Layer 0 --> 5
   Number of Neurons Layer 1 --> 10
   Number of Neurons Layer 2 --> 1
   Number of patterns --> 25108
   ******************************************************
   For net # 1 the mean of error is 11.304664 percent //error average
~11%
   For net # 1 the minimum error is 0.000260 percent //minimum error
   For net # 1 the maximum error is 56.434623 percent //maximum error
   24813 of 25108 error values are less of 37.04 percent (98.83) //99%
of the error is smaller than 37%
   ******************************************************
```

We can notice that the regulated error is 11% and that 99% of the errors are smaller than 37%. This value will be used to calculate RTO from RTT, when we adapt NN to ns2.

2.4 Interfacing NN and ns2

The connection of the NN model to ns2 is done so that NN gives the RTO estimation. This is necessary only for the flow with id=200, that is established between the nodes 0 and 3. JAC continues to regulate all the others flows. The NN is composed of functions in C++ that we adapt suitably to the ns2 code. During the simulation process, the NN estimates when there is a need for a RTT value according to the first previous RTT values (5 inputs values) produced by it.
From this value, RTO is produced as follows:

$$RTO=RTT*ERROR\_COR \quad (12)$$

where ERROR_COR=1+Upper_Limit, the Upper_Limit is the 99% percentile of the error values we calculated in the NN training control process.

Connecting this model with the previous results from the NN training Upper_limit=0.37, we get:

$$RTO=RTT*1.37 \quad (13)$$

After we successfully connected NN to ns2, we started the simulations with various scenarios as described in the next section.

3 Comparison of NN to JAC – Simulation Scenarios, Results

After interfacing NN with ns2 we are ready to start comparing NN to JAC by running different simulation scenarios. Specifically, we have two cases: the original one, where all flows follow the JAC algorithm, and the experimental one, where one flow follows the NN approach and the rest stick to JAC. We created various scenarios concerning the topology in the networks as we have presented it in the previous chapter and we monitored mainly the FTP flow with id=200. The source is node 0 and the destination the node3. We ran each scenario successively, first with ns2 and NN and then with the original ns2. NN and JAC ran always with a different and random initialization at the simulation parameters. We gathered the simulation results and counted the number of packets that were transmitted by the flow with id=200, the total retransmissions, and the correct and bad retransmissions. We also calculated the final throughput for each flow of the network. After we gathered these results we found the average and compared the NN results with JAC's.

In the following subsections we will present all the simulation scenarios and the results from the comparison. In brief, we can say that the NN is significantly more efficient than JAC, with a small penalty in unnecessary retransmissions.

3.1 1st Scenario: NN Against Training Scenario

In this scenario, we use the same scenario what we have used for the gathering of RTT values for the NN training. We have run simulations with simulation time 20000 sec. Each simulation is run 10 times successively for each method starting with ns2 adapted to NN. That is to say the first simulation has run with ns2 and NN. The second successive simulation has run with original ns2 and so on. To compare the simulations we took a pair of them, that is the first with the second the third with the fourth and so on. Afterwards we took the average of the simulation tests. We can see clearly in figures 3, 4 and 5 that the NN is more efficient from JAC and the cost in packets from incorrect retransmissions is low compared with the gain in terms of total number of transmitted packets. In addition we calculate final throughputs for all flows and we can see clearly that NN work does not pressure the others flows.

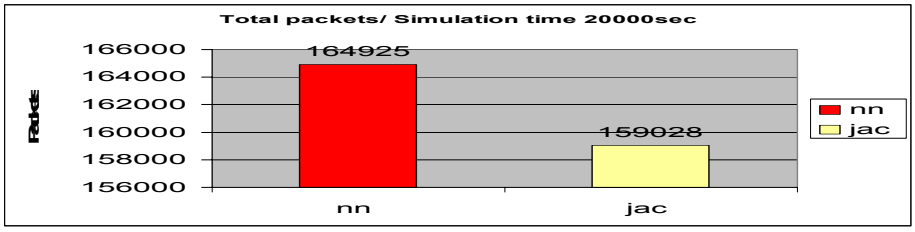

Fig. 3. Results of total packets – Training scenario.

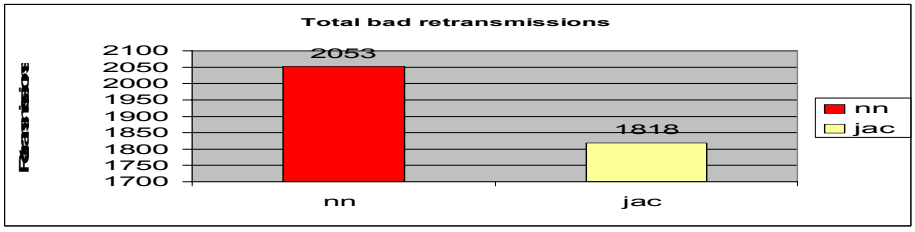

Fig. 4. Results of total bad (no useful) retransmission – training scenario

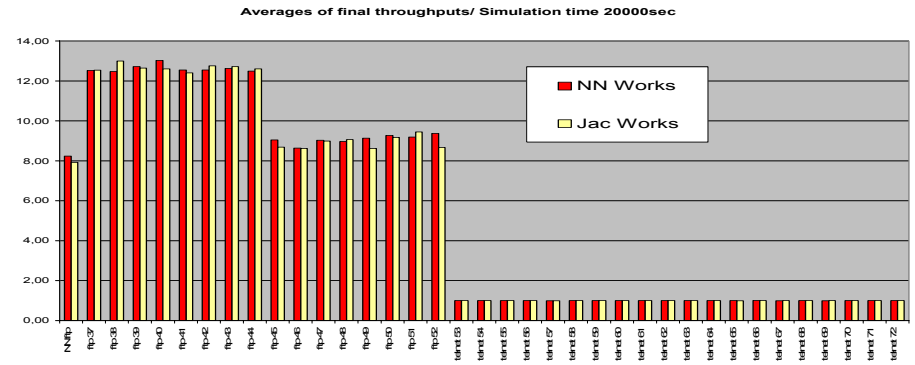

Fig. 5. Throughputs vs flows – Training scenario

3.2 2nd Scenario: Sudden Appearance of a Flow

From the above results, we have noticed that the NN performs well in scenarios where the various flows are the same as during training. But what happens, if suddenly a new flow appears to the network during the simulation process? To get an answer we introduced an FTP flow between nodes 5 and 11 that did not exist during the training process. Naturally there is a big changing and consequently NN has to deal with different situations. So its training will not probably be completely suitable. Here the results are separated in two time periods: before and after the appearance of the unexpected flow. Total simulation time is 20000sec. The critical point is 9000 sec. We notice that NN loses part of its efficiency but still continues to surpass JAC. In addition final throughputs before the critical points show that NN creates a little pressure to others flows specially the flows with id=45, 46, 47, 48, 49, 50 which have opposite direction of our flow(NN flow) direction. We don't see this phenomenon in results after the critical point (Figures 6, 7, 8 and 9)

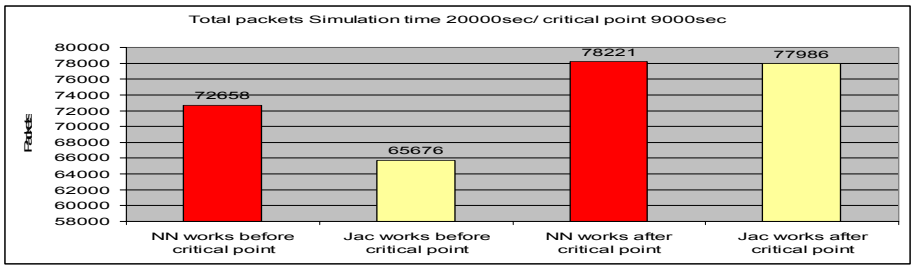

Fig. 6. Results of total packets before and after critical point (9000 sec)

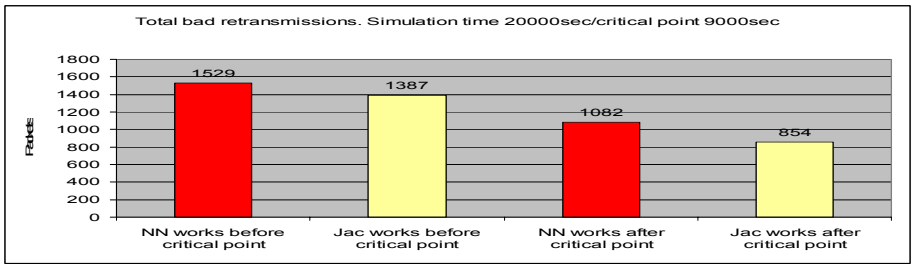

Fig. 7. Results of total bad (no useful) retransmissions before and after critical point (9000 sec)

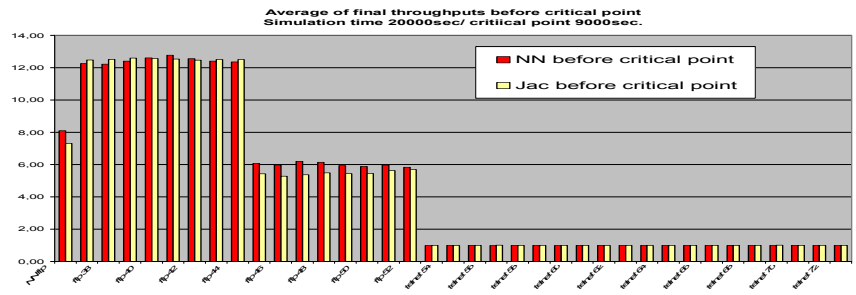

Fig. 8. Final throughputs before critical point (9000 sec)

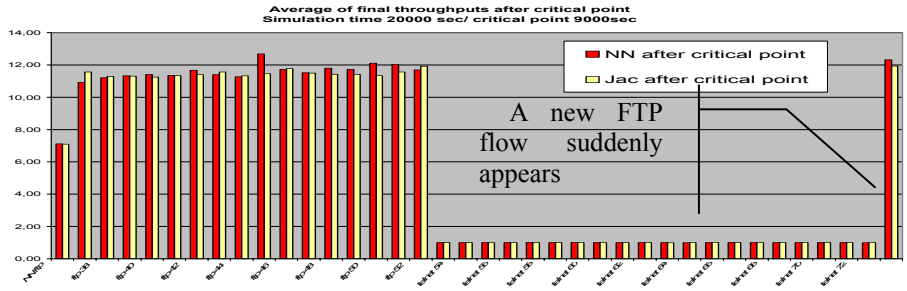

Fig. 9. Final throughputs after critical point (9000 sec)

3.3 3rd Scenario: Four New Flows Suddenly Appear

This is the most difficult situation. Four new FTP flows appears at the same time in the network. One in the nodes 5 and 11, one in the nodes 5 and 10, one in the nodes 6 and 9 and one in nodes 4 and 8. NN did it well with the one unexpected flow. The results show (figures 10, 11, 12 and 13) that NN is still doing very well in this difficult situation. The simulation time is 25000 sec and critical point is 10000 sec. In addition we calculate final throughputs for all flows and we can see clearly that NN work does nor pressure the others flows.

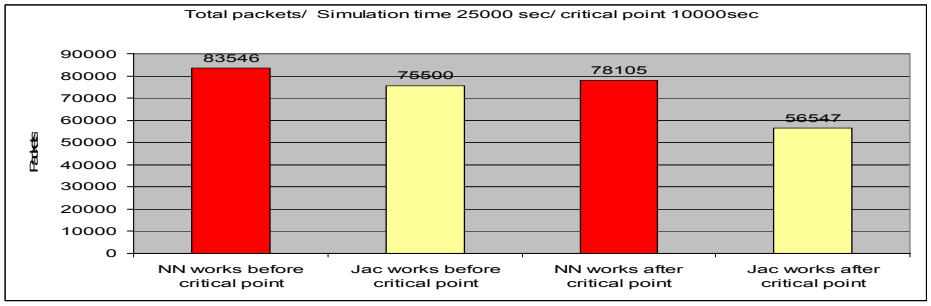

Fig. 10. Results of total packets before and after critical point (10000 sec)

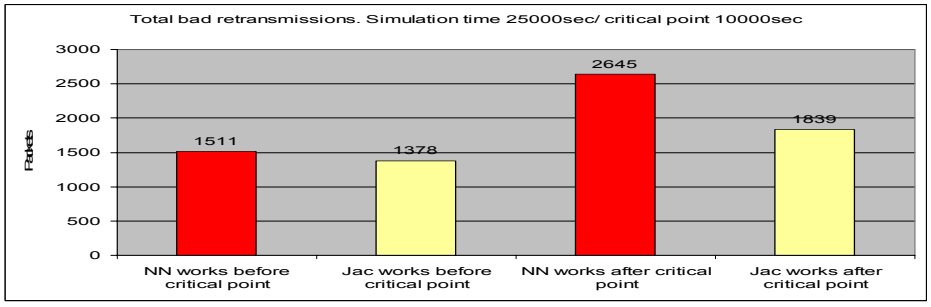

Fig. 11. Results of total bad retransmissions before and after critical point (10000sec)

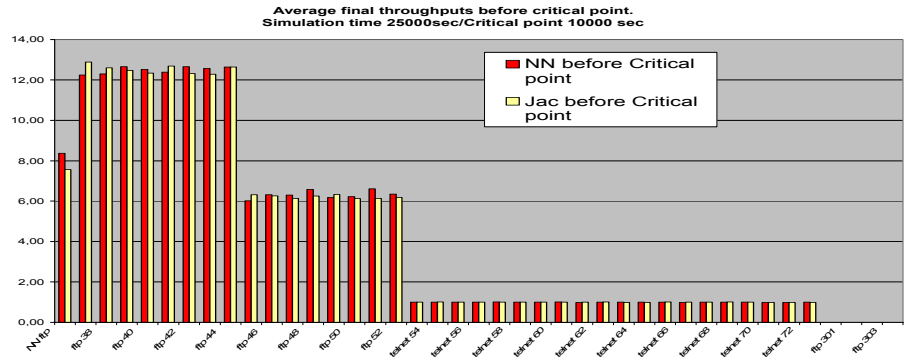

Fig. 12. Results of final throughputs before critical point (10000 sec)

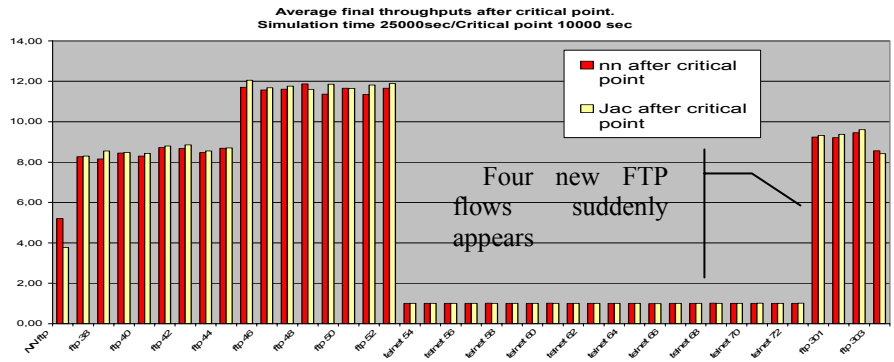

Fig. 13. Results of final throughputs after critical point (10000 sec)

4 Conclusions

In this paper we presented a neural network methodology for estimation purposes in the context of congestion control in TCP network. The obtained results confirm that NNs can constitute a reliable alternative for estimation of RTT and RTO in operation of TCP/IP. The cost from the useless retransmissions is compensated by the large gain from the transmission of a higher number of data packets from NN, always in comparisons with Jacobson's algorithm.

Further work will incorporate a more complicated topology of the simulated network. NN must also be tested in a network where it controls by itself the whole traffic.

References

[1] V. Jacobson, M. Karels. Congestion Avoidance and Control. *Proceeding ACM SIGCOMM Conference*, Stanford CA, (1988)

[2] P. Karn, C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocol. *Proceedings ACM SIGCOMM Conference*, (1987)
[3] Danzing J., tcplib:A Library of TCP Internetwork Traffic Characteristics. (1991) http://irl.eecs.umich.edu/jamin/papers/tclib/tcptiptr.pz.Z
[4] Danzig J., C Mitzel, D. Estrin, An Empirical Workload Model for Driving Wide-Area TCP/IP Network Simulations, *Internetworking: Research and Experience*, 3:1-26, (1993)
[5] J-C. Bolot. End-to-End Packet Delay and Loss Behavior in the Internet. *Proceeding ACM SIGCOMM Conference*, (1993)
[6] S. Haykin, *Neural Networks: a Comprehensive Foundation*, Prentice Hall, Upper Saddle River, NJ, (1999)
[7] G.P. Zhang, "An Investigation of Neural Network for Lineal Time-series Forecasting", *Computers and Operations Research*, 28(12):1183-1202, (2001)
[8] W. Stallings, *High Speed Networks TCP/IP Design Principles*, Prentice Hall, Upper Saddle River, NJ, (1998)
[9] C. Douligeris, A. Pitsillides, D. Panno, Computational Intelligence Techniques in Computer Networks, *Computer Communications*, 25(26) (2002)
[10] K. Fall, K. Varadhan "The ns Manual", http://www.isi.edu/nsnam/ns/ns-documentation
[11] RFC: 793, Transmission Control Protocol, (1981), http://www.ietf.org/rfc/rfc0793.txt
[12] T.M. Peng, A.D. Papalexopoulos, On the Nonlinear Properties of Feedforward Neural Networks, *Engineering Intelligent Systems*, 2:67-73, (1996)