Using BP Neural Network for Adapting Playout Time in Communication Networks

Sara Helmi[1, 2], Niveen Mohamed Badra[2], Mohamed Elkattan[3]

[1]Department of Basic Science, Faculty of Engineering, British University in Egypt, Cairo, Egypt
e-mail: Sara.helmy@bue.edu.eg
[2]Department of Engineering Physics and Mathematics, Faculty of Engineering, Ain Shams University, Cairo, Egypt
e-mail: niveen_badra@eng.asu.edu.eg
[3]Nuclear Materials Authority, Cairo, Egypt
e-mail: emtiazegf@hotmail.com

***Abstract*-New multimedia applications have a critical requirement on jitter. Network jitter is a serious problem in communication networks, especially in Voice over IP networks (VOIP). One of the proposed solutions to minimize jitter is to adapt the playout time. In our paper, we introduce an adaptive approach using Back Propagation (BP) neural network to identify the jitter and adjust the playout time according to several network conditions. The algorithm was tested using k-fold cross validation and the results show that the algorithm can achieve promising results under different delay conditions.**

Keywords-Jitter, playout time, optimization, Neural Network, Back-propagation neural network, k-fold cross validation

I. INTRODUCTION

In VOIP communication, several requirements for Quality of Service (QOS) affect the speech quality [1][2]. There are three main performance aspects that characterize the quality of voice in communication networks over the Internet. The first aspect is the end-to-end delay, which is the time it takes for a packet to be transmitted across a network from source to destination. Acceptable end-to-end delay values are less than 100 ms for the one way delay [3][4]. The second aspect is the packet loss, as described in [5]. The acceptable range of the voice quality is when the packet loss is less than 2%. The last aspect is the delay jitter. It is defined as the difference in end-to-end one-way delay between selected packets in a flow with any lost packets being ignored.

Delay jitter is the result of network congestion and improper queuing is the delay jitter. In the sending host, the voice packets are transmitted at a steady rate, but packets are received at a disparate rate [6]. To be able to recover the initial steady rate, the played out packets should be at a steady rate. When the jitter is large, dropping of the delayed packets can occur, and that will lead to obvious audible gaps. The sense of hearing in humans is highly sensitive for short audio gaps. That is the reason behind keeping the jitter in to a minimum value (less than 30 ms). Other elements mentioned in [7] state that the acceptable jitter can be between 30 ms and 75 ms.

In VOIP applications, the mechanism that is used to make the rate of output packets constant is the play out buffer, also known as the jitter buffer. The jitter buffer holds the late packets and then plays them out at a steady rate. There are two kinds of jitter buffers: static buffers and dynamic buffers [8]. An adaptive jitter buffer is the most applicable mechanism, because it uses several strategies in the sender and the receiver hosts. When network conditions are perfect (the variation in interspace delay is almost 0 ms), it adjusts to be at a minimum value to reduce latency. On the other hand, when network conditions are very hard (a high transient jitter and packet loss exist), it adjusts itself to a higher value. However, this has the side effect of increasing the latency [9].

Many other works have been proposed to improve jitter. In [10], Artificial Neural Network (ANN) was combined with a standard multi-layer perceptron (MLP) and a recurrent-MLP [11] and Wavelet Packet-MLP (WP-MLP) [12]. Our new approach is combining ANN with the algorithms Exponential-Average, Fast-Exponential-Average and Min-Delay [13]. We use back propagation (bp) neural network on the three algorithms and we train the (bp) neural network to choose which algorithm is suitable to be run in different network conditions. This offered a flexibility in the network and a wide range of conditions that the network can deal with. We will focus on optimization of the playout time to achieve the best trade-off between latency and dropping of a voice packet.

The following sub-section presents the background for the problem formulation in Section II. It is the basic concept behind sending packets with time i from a sender to a receiver [13]. Fig. 1 shows a schematic diagram that illustrates the packet timing from sender to receiver. Table 1 shows the description of each parameter in "Fig. 1".

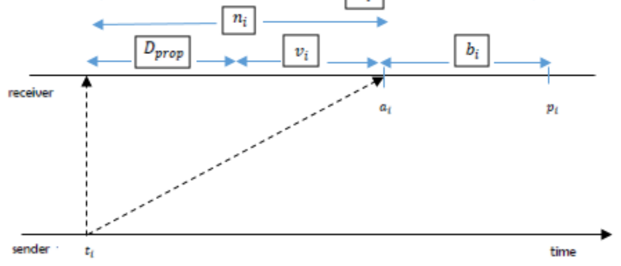

Figure 1. Packet ***i*** timing from sender to receiver [13]

TABLE 1. NOTATION PACKET ***i*** TIMING

Notation	Description
t_i	The time of sending *i*th packet.
a_i	The received time of packet ***i*** at the receiver. $a_i = t_i + n_i$
p_i	The played out time for packet ***i*** at the receiver.
Dprop	The delay of propagation from the sender to the receiver.
v_i	The delay of packet ***i*** from the sent to the destination.
b_i	buffer delay for packet ***i*** $b_i = p_i - a_i$
d_i	The time elapsed from transmitting from the source until it is played out at destination, (playout delay)**.**
n_i	Packet network delay. $n_i = D_{prop} + \widehat{v_i}$

In [13], three adaptive playout delay adjustment algorithms were defined:

- Algorithm 1 (Exponential-Average): This algorithm uses the mean $\widehat{d_i}$ to estimate the playout delay $\widehat{p_i}$ of its arriving packet so that:

$$\widehat{d_i} = a * \hat{d}_{i-1} + (1-a) * n_i \qquad (1)$$

- Algorithm 2 (Fast-Exponential-Average): The only difference between this algorithm and the previous one is the new condition when the network delay n_i is larger than $\hat{d}_{i-1}$, β was chosen as 0.75 as in [13].

```
if $(n_i > \widehat{d_i})$                    (2)
    $\widehat{d_i} = B * \hat{d}_i + (1-B) * n_i$
else
    $\widehat{d_i} = a * \hat{d}_i + (1-a) * n_i$
end
```

- Algorithm 3 (Min-Delay): The proposal of this algorithm is to minimize the delays. It uses the minimum delay of all packets received in the current talkspurt as $\widehat{d_i}$ to predict the next talkspurt playout delay. Let $\widehat{S_i}$ be the set of all packets received in a single talkspurt $\widehat{d_i} = min_{j \in S_i}\{n_i\}$

```
if ($n_i$ < $n_{i-1}$)                    (3)
    $\hat{d}_i = n_i$
else
    $\hat{d}_i = n_{i-1}$
End
```

The remainder of this paper is structured as follows. Section II provides a description and overview of the problem formulation. Section III provides our modeling approaches. Section IV provides the optimization approach wit back propagation (NN). Section V presents the validation test using k-fold cross validation and the results, and, finally, Section VI gives the conclusions.

II. PROBLEM FORMULATION

If the first packet ***i*** in a talkspurt is played out, then its calculation will be as below, where $\widehat{d_i}$ is the mean and $\widehat{v_i}$ is the variation in the end-to-end delay:

$$p_i = t_i + \widehat{d_i} + \mu * \widehat{v_i} \qquad (4)$$

In this paper, we use a uniform distribution for random variance $\widehat{v_i}$ from 0 to 10 in normal conditions because the equation of variance [13] cancels the delay in the network, which is not realistic in a real network communication. A certain value 0.998002 of the weighting factor α was mentioned in [13] and, after applying some statistical analysis with small differences from the specific value 0.998002, α was settled as a value varying between 0.01 ≤ α ≤ 0.998002. The playout time for any posterior packet j in a talk spurt is computed as:

$$p_j = p_i + t_j - t_i \qquad (5)$$

The term μ in the playout time is used to keep the playout time beyond the delay estimate so that only a small number of the packets in the receiver will be lost due to late arrivals [14]. In this paper, we set a range of values 1 ≤ μ ≤. 20. Also, we considered that, if the packet ***i*** arrived at the receiving host at the same time or after its playout time, the packet ***i*** will be played out:

```
if ($p_i$ <= $a_i$)
(6)
    $p_i = a_i$
```

TABLE 2. NORMAL CONDITIONS OF NETWORKS ($\times 10^{-15}$)

α	0.01		0.26		0.51		0.76		0.875		0.9		0.925		0.95		0.998002	
d_1	μ = 10	7.4	μ = 12	8.9	μ = 6	7.3	μ = 14	8.3	μ = 12	8.3	μ = 1	9.1	μ = 17	8.4	μ = 3	9.7	μ = 1	8.4
d_2	μ = 4	8.3	μ = 9	9.4	μ = 15	9.1	μ = 6	7.4	μ = 18	8.4	μ = 18	8.3	μ = 7	8.7	μ = 2	9.1	μ = 9	7.2
d_3	μ = 14	7.6	μ = 6	9.6	μ = 18	7.6	μ = 16	7.4	μ = 12	8.3	μ = 16	8.9	μ = 6	8.6	μ = 6	7.4	μ = 2	8.7
min	μ = 10	7.4	μ = 12	8.9	μ = 6	7.3	μ = 6	7.4	μ = 12	8.3	μ = 18	8.3	μ = 17	8.4	μ = 6	7.4	μ = 9	7.2
	d1		d1		d1		d2		d1		d2		d1		d3		d2	

A new formula is proposed to calculate the old jitter at a_i by using:

$$|a_j - a_1| - NT \qquad (7)$$

And the new jitter at p_i by using

$$|p_j - p_1| - NT \qquad (8)$$

N: 1, 2, 3, 4…

We assumed that the packets will be sent at a constant rate. So, T refers to a constant time for sending the packets at the sending host, which will be 1.2.

To evaluate the best parameters for different network conditions, we set 100 packets with eight conversations, each conversation including 5 to 20 packets. Under normal condition of the network where variance changes from 0 to 10, and the initial value for the playout delay d_0 will be 29, and propagation delay 20. The mean and standard deviation for the proposed formula (8) have been calculated in the receiving host. After that, we evaluate the performance of the three algorithms d_1, d_2, d_3 in normal conditions. With variance (v), propagation delay (D_{prop}) and initial value where used, for each α value we changed μ from 1 to 20. With total simulation trials of 27, and after conducting those trials we choose the minimum mean for the calculated function and identify the corresponding μ to it.

III. IMPLEMENTATION

Our approach for changing the network conditions is summarized in five stages, as follows:

Stage 1. Comparing the results of the three adaptive algorithms under normal conditions (i.e. d_1, d_2, d_3 respectively) and selecting the minimum jitter value in the receiving host with the corresponding μ (see Table 2). The last row in Table 2 shows the selected algorithm for each set of trials that gives the minimum playout value between the three algorithms while comparing over a certain α.

Stage 2. Repeating this process for the three algorithms will be done now under different network conditions, first, by varying the variance from 0 to 15, 0 to 20, and 0 to 25 respectively. This leads us to a really hard condition in queuing delay. Different results are shown for algorithms that give the minimum values.

Stage 3. Inserting a different value for the propagation delay 40, 60, and 80 to create a new set of network simulations.

Stage 4. Varying the initial value for playout delay d_0 from 50, 75, and 100.

Stage 5. In the final stage, we vary the variance v, mean d_0 and propagation delay D_{prop} simultaneously to evaluate the algorithms in terms of achieving the minimum jitter in the receiving host under rough conditions. The previous five stages results are summarized in Table 3.

TABLE 3. INPUTS AND OUTPUTS RESULTS

No. training set	Inputs			Outputs		
	v	D_{prop}	d_0	α	μ	Algorithm
1	0 to 10	20	29	0.998002	9	d2
2	0 to 15	20	29	0.875	18	d2
3	0 to 20	20	29	0.9	18	d2
4	0 to 25	20	29	0.26	6	d3
5	0 to 10	40	29	0.01	4	d2
6	0 to 10	60	29	0.01	4	d2
7	0 to 10	80	29	0.9	16	d3
8	0 to 10	20	50	0.76	16	d3
9	0 to 10	20	75	0.76	16	d3
10	0 to 10	20	100	0.76	16	d3
11	0 to 15	40	29	0.925	7	d2
12	0 to 15	60	29	0.9	16	d3
13	0 to 15	80	29	0.51	18	d3
14	0 to 20	40	29	0.998002	9	d2
15	0 to 20	60	29	0.925	7	d2
16	0 to 20	80	29	0.998002	2	d3
17	0 to 25	40	29	0.51	6	d1
18	0 to 25	60	29	0.76	14	d1
19	0 to 25	80	29	0.998002	2	d2

IV. Optimization Approach

Neural networks are modeled on the mechanism of the brain [Kohen, 1989] [Hecht-Nielsen, 1990]. Neural networks can perform various duties like classification, identification, pattern recognition, control systems, speech and vision.

A supervised learning algorithm adjusts the strengths or weights of the inter-neuron connections according to the difference between the desired and actual outputs corresponding to a given input. Thus, supervised learning requires a "teacher" or "supervisor" to provide desired or target output signals.

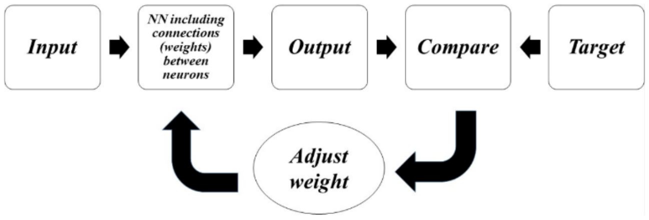

Figure 2. Training a neural network

The main objective of neural networks is to adjust a particular input to lead to a specific target output. This situation is illustrated in Fig. 2. A comparison has been adjusted in the network between outputs and targets, until the output matched the target in the network. In the proposed model, back propagation neural network (Rumelhart and McClelland, 1986) was used in Layered feed-forward ANNs, as in [15]. This means that the artificial neurons are organized in layers and send their signals "forward", and then the errors are propagated backwards. The network receives inputs by neurons in the input layer, and the output of the network is given by the neurons on an output layer. There may be one or more intermediate hidden layers, as in [15].

Neural Network Implementation

In our approach, we first conduct linear scaling of data (i.e., data normalization), which is important to put the data in interval between zero and one. This requires $mean$ and $standard\ deviation$ values associated with the facts for a single data input$x = (x - mean)\ \ (standard\ deviation)$, and then the error (difference between actual and expected results) is calculated.

Our training data set in Table 3 consists of 19 input signals assigned with corresponding target (desired output). The neural network is then trained using back propagation algorithms. We separate the three desired outputs to construct three separate networks. We train the network with the 19 training data in each round a three input from , d_0, D_{prop} will inter the three input neuron. Each BP neural network consists of input layer with three neurons, one hidden layer with five neurons, and output layer consist of one or three outputs based on the type of the network, two of them with only one output and that will be for outputs α and μ in Table 3; see Fig. 3.

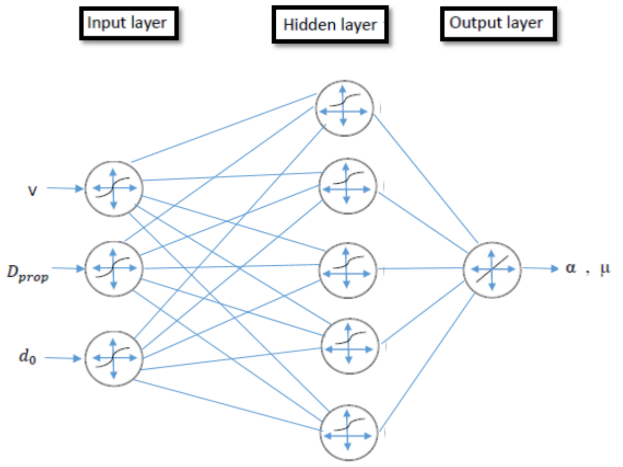

Figure 3. BP neural network for outputs α and μ

For the input layer, we use hyperbolic tangent sigmoid as a transfer function, which varies in the interval [-1, 1]. For the hidden layer, we used Log-sigmoid transfer functions to calculate a layer's output in the interval [0, 1] from its net input. At the output layer, we used linear transfer function. Fig. 4 and Fig. 5 show the performance of the two BP neural networks after 1000 epoch. It can be seen that the curve converges to a minimum value after 1000 epoch.

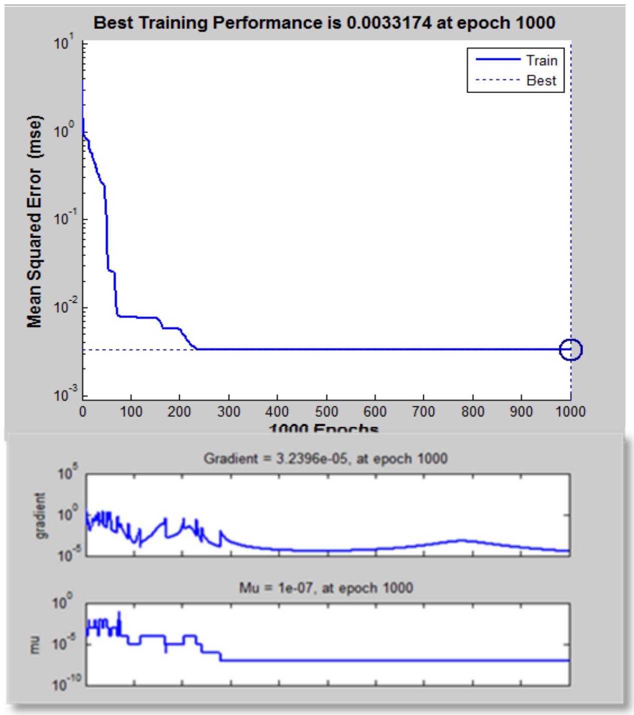

Figure 4. Performance of α

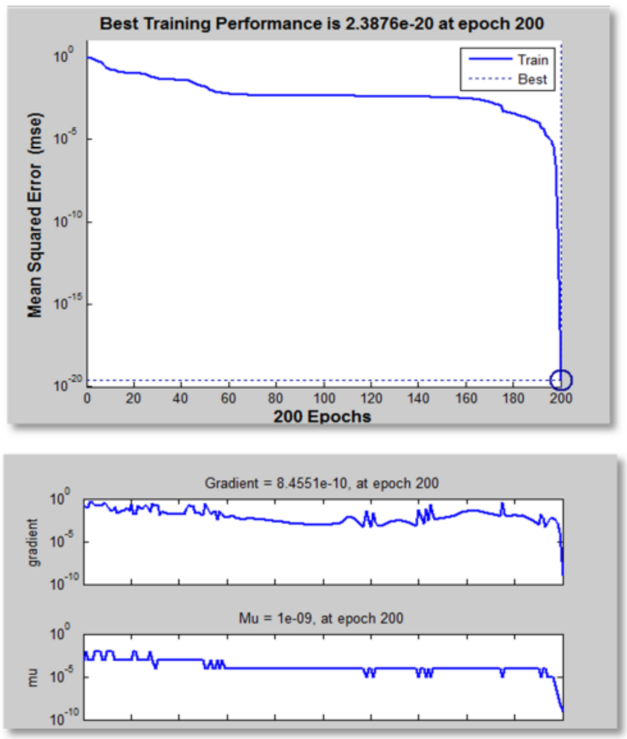

Figure 5. Performance of μ

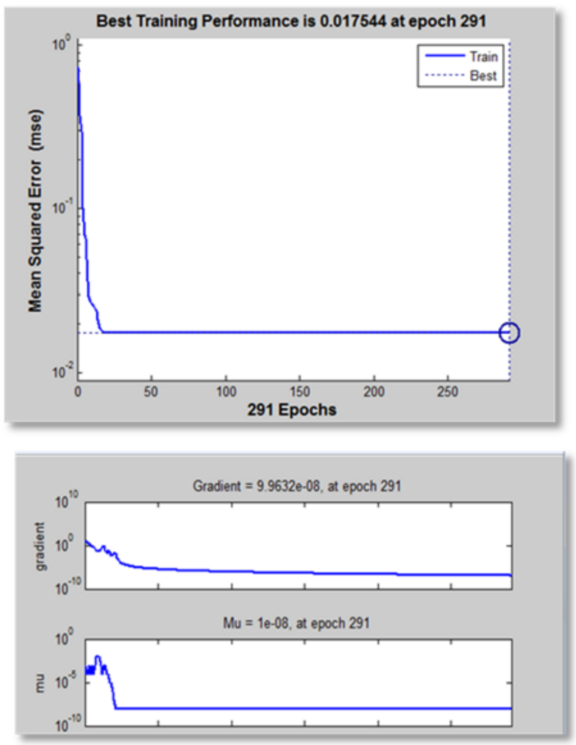

Figure 7. Performance of network algorithm

Classification

A classification problem occurs when an object needs to be assigned into a pre-defined group or class based on a number of observed attributes related to that object as in [16] [17]. Neural networks (NNs) are popular classification algorithms in computer-aided diagnosis because of their ability to ‘‘learn’’ classification rules from a set of training data (see Fig. 6).

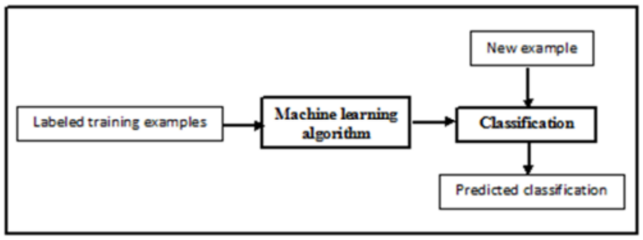

Figure 6. Classification in machine learning

The network with output algorithm is used to calculate the playout time related to the classification problem. We used a three-output network to sense the classification in d_1, d_2, d_3. Fig. 7 shows the performance out of 1000 epoch. It can be seen that neural network succeeds in classifying each one to the appropriate algorithms.

Now, let us test and evaluate our model using the *k*-fold cross validation method. The test will give us a really good overview of the model performance.

V. *K*-FOLD CROSS VALIDATION

To be able to build a strong and useful machine learning solution, we need suitable analytical tools for evaluating the performance of our system. We can calculate the prediction errors (differences between the actual response values and the predictions) and summarize the predictive ability of the model by the mean squared prediction error (MSPE)**.** This gives an indication of how well the model will predict in the future. Sometimes. the MSPE is rescaled to provide a cross-validation R^2. However, most of the time. we cannot obtain new independent data to validate our model. An alternative is to partition the sample data into a training (or model-building) set, which we can use to develop the model, and a validation (or prediction) set**,** which is used to evaluate the predictive ability of the model. This is called cross-validation.

Here, we will clarify the reasons why *k*-fold cross validation is used in neural networks. The main goal of the classification problem is to find a group of weights and bias values that will lead to a perfect match between the targets and the output values, as in [18] [19].

Our approach would use all of the 19 data items to train the neural network. However, this approach would find weights and bias values that match the target value extremely, in fact, probably with 100 % accuracy, but when presented with a previously unseen set of input data, the neural network would fail to predict well. This case is called over-fitting. To overcome this problem (over-fitting), we use the *k*-fold cross validation. The idea behind *k*-fold cross validation is to randomly sort your data and divide your data into *k* equally-sized sets, unless the data cannot be divided equally, like in our example. Each set is used one time as the test set while the rest of the data is used as the training set. The learner trains for *k* rounds, each round using one of the sets as the validation set and the remaining sets as the training set. We measure its accuracy on the validation set. Then, we average the accuracy over the *k* rounds to get a final cross validation accuracy, as in [20]. We used a 4-fold cross validation three folds with size 4 and the last fold sized 7, as shown in Fig. 8.

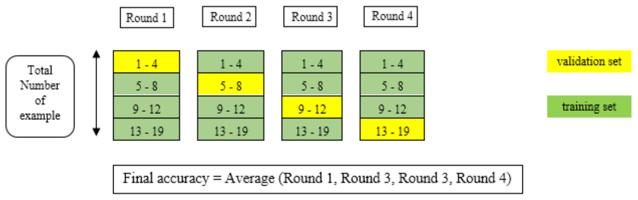

Figure 8. 4-fold cross validation

By using $E = \frac{\sum_{I=1}^{k} n_i}{m}$ (9)

where,

E	Accuracy for the neural network.
n_i	The number of examples in Fold *i* that were Correctly classified.
m	Number of examples which is equal 19.

the calculated accuracy for the neural network with output µ is 0.26789 and for the neural network with output α is 0.315789; the neural network accuracy for the output algorithm is 0.263157.

Repeated k-fold cross validation

It will not be effective to take the mean of 4 samples. On the other hand, splitting our sample into more than 4 would greatly reduce the stability of the estimates from each cross validation. A way around this is to do repeated *k*-folds cross validation. To do this, we simply repeat the *k*-fold cross validation three times, each time with a different random arrangement and take the mean of this estimate. An advantage of this approach is that we can also get an estimate of the precision of this out-of-sample accuracy by creating a confidence interval. We will do three replications so we end up with a nice round: 12 out-of-sample accuracy estimates. By using the following formulas:

$$e = \frac{\sum_{j=1}^{t} E_j}{t}\ ,\quad v = \frac{\sum_{j=1}^{t}(E_j - e)^2}{t-1}\ ,\quad \sigma = \sqrt{v} \qquad (10)$$

where,

e	Mean of the three runs for each one of the three networks.
$E_1 \ldots E_t$	Accuracy estimates obtained in t runs, which is in our case 3 runs.
v	Variance (variability it shows for different samples)
σ	Standard deviation (How much it deviates from the true value).

This time, we get an estimate for network µ, α and algorithm, which is quite close to our estimate from a single *k*-fold cross validation. We obtained the mean, variance, and standard deviation of the 3 runs for network µ, α and algorithm in Table 4.

TABLE 4. ACCUARACY IN THREE REPLICATIONS

Neural Network	**Mean (e)**	**variance (v)**	**Standard deviation (σ)**
Network µ	0.0877	6.463×10^{-3}	0.080393
Network α	0.3333	6.463×10^{-3}	0.080395
Network Algorithm	0.3157	2.769×10^{-3}	0.0526

The standard procedure for training a neural network involves training on the complete database by minimizing the accumulated misclassification of inputs in the dataset. Since the overall goal is not to minimize errors on the dataset, but rather to minimize misclassification on a much larger set of conceivable inputs, cross- validation gives a much better measure of expected ability to generalize. The estimate for the algorithm performance has an error of 0.0877 in network µ, 0.3333 in network α and 0.3157 in network algorithm, with standard-deviation of 0.080393 in network µ, 0.080395 in network α and 0.0526 in network algorithm. The validation error gives an unbiased estimate of the predictive power of a model. So, after comparing the mean of the error (e) of the three networks with the actual error from training the 19 data sets in the network, we conclude that more training samples need to be used to obtain better results and provide a good network performance, and this can be achieved by monitoring a real voice over IP network.

VI. CONCLUSION

In this paper, we have investigated the performance of our algorithm under different conditions of the communication network, starting from the normal conditions to congested conditions which have high queuing, propagation delay, and playout delay for adapting the buffering of packets at the receiver. The main objective is to build an algorithm that can transact with all these conditions and to keep the playout delay as small as possible. We proposed an optimization method for optimizing the playout delay of packets. Using back propagation neural network, the results were promising in dealing with the input data. In our future work, we will conduct further investigation of more realistic conditions, such as larger data sets with more complicated distributions.

References

[1] J. Zhang, H. J. Kim and D. H. Ahn, "Analysis of Streaming Service Quality Using Data Analytics of Network Parameters, " Proc. 2012 DATA ANALYTICS 2012 : The First International Conference on Data Analytics

[2] O. Obafemi, T. Gyires and Y. Tang, " An Analytic and Experimental Study on the Impact of Jitter Playout Buffer on the E-model in VoIP Quality Measurement , " Proc. 2011 ICN 2011 :The Tenth International Conference on Networks.

[3] E. TIPHON, "End-to-End Quality of Service in TIPHON Systems,"Proc. 2000 Definition of Quality of Service (QoS) Classes,Vols. Part 2: 101 329-2.

[4] Dr. H. A. Mohammed, Dr. A. H. Ali and H. J. Mohammed, "The Affects of Different Queuing Algorithms within the Router on QoS VoIP application Using OPNET," Proc. 2013 International Journal of Computer Networks & Communications (IJCNC) Vol.5, No.1

[5] T. Uhl, "QUALITY of service in VOIP communication," Proc. 2004 Int.J.Electron.Commun 58 (3), pp. 178-182.

[6] H. Dahmouni, A. Girard and B. Sansò, "An analytical model for jitter in IP networks," Ann. Telecommun. Proc. 2012 67:81–90.

[7] E. DTR /TIPHON, "Telecommunication and Internet Protocol Harmonization Over Networks (TIPHON)," General Aspects of Quality of Service (QOS), Proc. 1998 vol. tr 101 329 , no. Ver. 1.2.5

[8] R. J. B. Reynolds, A. W. Rix, 2001"Quality VOIP - an engineering challenge," Proc. 2001 BT Technol Journal Vol.19 Issue:2 pp. 23-32.

[9] I. Klimek, M. Čajkovský and F.Jakab, "Novel methods of utilizing Jitter for Network Congestion Control," Proc. 2013 Acta Informatica Pragensia, 2(2), 1–24, DOI: 10.18267/j.aip.20

[10] Y. Zhang, D. Fay and L. Kilmartin, "An application of neural networks to adaptive playout delay in VoIP," Proc. 2007 Ireland Conference on Information and Communication Technologies

[11] S. Haykin, "Neural Networks: A Comprehensive Foundation," Proc. 1998 Prentice. Hall, Upper Saddle River, NJ. ISBN:0132733501

[12] Q. Zhang and A. Benveniste, Wavelet networks. IEEE Trans. Neural Networks, Proc. 1992 3:889–898

[13] R. Ramjee, J. Kurose, D. Towsley and H. Schulzrinne, "Adaptive playout mechanism for packetized audio application in wide-area netwoks," Proc. 1994 Proseeding of IEEE infocom, vol. 2, no. ISBN: 0-8186-5570-4, pp. 680-688.

[14] V. Jacobson, "Congestion avoidance and control," Proc.1988 ACM SIGCOMM Conf Stanford, pp. 314-329, August, 1988.

[15] Z. Reitermanoy, "Feedforward Neural Networks - Architecture Optimization and Knowledge Extraction," Proc.2008 WDS'08 Proceedings of Contriuted Papers, Part 1, pp. 159-164.

[16] R. P. Lippmann, "Pattern classification using neural networks," Proc. 1989 IEEE Communications Magazine, vol.27, no. Issue: 11.

[17] Y. L. Murphey and G. Ou, "Multiclass pattern classification using neural networks," Proc. 2007 science direct journal Pattern Recognition, vol.40, Issue:1 pp. 4-18.

[18] S. Haykin, "Neural Network sand Learning machines,"Proc. 2009.

[19] A. Perez, J. A. Lozano and J. D. Rodriguez, "Sensitivity Analysis of K-Fold Cross Validation in Prediction Error Estimation," Proc.2010 IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.32, Issue: 3 pp. 569-575.

[20] T. Fushiki, "Estimation of prediction error by using K-fold cross-validation," Proc. 2009 Springer Science Business Media, LLC 2009.