A Perfect QoS Routing Algorithm for Finding the Best Path for Dynamic Networks

Hazem M. El-Bakry

Faculty of Computer Science & Information Systems,
Mansoura University, EGYPT
helbakry20@yahoo.com

Nikos Mastorakis

Dept. of Computer Science
Military Institutions of University Education
(MIUE) - Hellenic Academy, Greece

Abstract :- It is highly desirable to protect data traffic from unexpected changes as well as provide effective network utilization in the internetworking era. In this paper the QoS management issue that utilizing the active network technology is discussed. Such algorithm is based on the proposed work presented in [15]. Active networks seem to be particularly useful in the context of QoS support. The Active QoS Routing (AQR) algorithm which is based on On-demand routing is implemented incorporating the product of available bit rate and delay for finding the best path for dynamic networks using the active network test bed ANTS. It is inferred that with background traffic, the AQR finds alternative paths very quickly and the delay and subsequently the jitter involved are reduced significantly. In this paper the variant of AQR implemented is demonstrated to be more useful in reducing the jitter when the overall traffic in the network is heavy and has useful application in finding effective QoS routing in ad-hoc networks as well as defending DDoS attacks by identifying the attack traffic path using QoS regulations. The main achievement of this paper is the fast attack detection algorithm. Such algorithm based on performing cross correlation in the frequency domain between data traffic and the input weights of fast time delay neural networks (FTDNNs). It is proved mathematically and practically that the number of computation steps required for the presented FTDNNs is less than that needed by conventional time delay neural networks (CTDNNs). Simulation results using MATLAB confirm the theoretical computations.

Keywords:- Routing Algorithm**,** Data Protection, Best Path, Dynamic Networks, Fast Attack Detection, Neural Networks

1 Introduction

The widespread growth of the Internet and the development of streaming applications directed the Internet society to focus on the design and development of architectures and protocols that would provide the requested level of Quality of Service (QoS) [15-26]. QoS is an intuitive concept defined as “the collective effect of the service performance which determines the degree of satisfaction of a user of the service” or “a measure of how good a service is, as presented to the user. It is expressed in user understandable language and manifests itself in a number of parameters, all of which have either subjective or objective values”.

The goal of AN is to support customized protocol mechanisms that can be introduced in a network. Differences in known AN approaches concern, e.g., the question of whether remote applications should be able to download protocol mechanisms to a node (router) or whether this right should be reserved to the operators of nodes, and the question of whether the code for these mechanisms should be carried as an additional payload by the data packets in transit or whether shipping and installing such code should be separated from the issue of data transfer. Simulation results of this paper are compared with the results presented in [15] and high improvement in the QoS parameter jitter is appreciated when applied on active network with slight modifications in the application of the proposed algorithm.

In addition, the main objective of this paper is to improve the speed of time delay neural networks for fast attack detection. The purpose is to

perform the testing process in the frequency domain instead of the time domain. This approach was successfully applied for sub-image detection using fast neural networks (FNNs) as proposed in [1,2,3]. Furthermore, it was used for fast face detection [7,9], and fast iris detection [8]. Another idea to further increase the speed of FNNs through image decomposition was suggested in [7]. FNNs for detecting a certain code in one dimensional serial stream of sequential data were described in [4,5]. Compared with conventional neural networks, FNNs based on cross correlation between the tested data and the input weights of neural networks in the frequency domain showed a significant reduction in the number of computation steps required for certain data detection [1,2,3,4,5,7,8,9,11,12]. Here, we make use of the previous theory on FNNs implemented in the frequency domain to increase the speed of time delay neural networks for fast attack detection.

2 QoS Routing

Active Networks (AN) is a framework where network elements, essentially routers and switches are programmable. Programs that are injected into the network are executed by the network elements to achieve higher flexibility and to present new capabilities. With the help of AN programs can be injected in to the network and executed with in the network itself without involving the end systems. QoS routing is a term used for routing mechanisms which consider QoS. It suffers from the static nature of networking today. QoS routing is bound to the use of common metrics and procedures which usually rely on distributed network performance data (increasing network traffic) and sophisticated algorithms (increasing the processor load on routers) but usually yield considerable improvements only for certain classes of applications. In [16], the authors used randomness at the link level to achieve balance between the safety rate and delay of the routing path.

A) QOS SUPPORT IN ACTIVE NETWORKS

Possible utilizations of AN to support QoS roughly fall into the following categories [15]:
1. Mechanisms which transfer application layer functionality into the network:
2. Mechanisms which are usually associated with layers 3 or 4: AQR mechanism falls in this category.
3. Mechanisms which rely on non-active QoS provisioning mechanisms: Here, AN are merely used to add greater flexibility to the specification of a QoS request.

B) AQR OPERATION

AQR is an on demand based QoS routing algorithm. The AQR algorithm can be described as follows as in [15].
1. The AQR sender calculates all non-cyclic paths to the destination from the link state routing table.
2. A probing packet carrying the QoS requirements, code for QoS calculation, the sender and receiver's addresses and a list of visited nodes is sent to each first hop of these paths.
3. Upon receiving an AQR probing packet, an AQR compliant transit node executes the AA code (contained in the packet or cached), which
· checks if the minimum QoS requirements found in the packet can be met (if a threshold say a maximum delay is exceeded, the packet is dropped),
· compares and updates the QoS data,
adds itself to the list of already visited nodes,
· Executes the code of the AQR sender, starting at step 2 — except that no probing packets are sent to the source or to any other already visited node (packets are multicast in the proper direction at each AQR-compliant transit node).
4. Only packets which conform to the minimum QoS requirements reach the AQR receiver, where a list of valid paths is generated. After a predefined period, the best path is chosen and communicated to the sender. If there is more than one best path, the traffic is split among the best paths.

C) DESIGN CONSIDERATIONS

The resources used are network bandwidth and CPU cycles to load the network. The system assumes overlay mode of deploying active code in network. The security issues are taken care of by the ANTS system itself For AQR, these parameters are delay and available bandwidth. For each of these parameters, a channel is given the properties of the underlying network .The overall flow is depicted as follows.

There are three important modules in the system. They are Application, Protocol and capsule modules. The application module consists of Router, Sender, Receiver sub modules. The capsule module consists of DataCapsule, Probecapsule and Replycapsule sub modules.

3 Implementation Details

The proposed AQR algorithm is implemented for the sample network topology shown below. In ANTS, the network topology is configured. The topology used is shown below. The routing protocol is implemented using the ANTS toolkit. ANTS is an EE running over the node OS Janos. The protocol is implemented as an active application that runs on each of the nodes of the network. In ANTS, there are three special classes to create capsules, protocols and active applications. The detailed information about ANTS is provided in [8].A new protocol is developed by sub classing the virtual class Protocol. This requires identifying all of the different types of packet that will enter the network by their different forwarding routines. Each type of packet and its forwarding routine is specified by sub classing the virtual class Capsule.
· A new application is developed by sub classing the virtual class Application.
· An instance of the class Node represents the local ants runtime.
· A new protocol and application are used by creating instances of their classes and attaching them to node. The application is connected to the node in order to send and receive capsules from the network.
· The protocol is registered with the node in order for the network to be able to obtain its code when it is needed.

This model allows customization by the network users assuming that a network of active nodes already exists and is up and running. The active nodes, however, will often be part of the system under study, particularly for experimentation with different topologies, application workloads, and node services. For these purposes, ANTS provides two facilities:

· Configuration tools allow a network topology complete with applications to be managed. This includes the calculation of routes and the initialization of local node configurations.

· A node extension architecture allows different nodes to support different service components, e.g., multicast, caching, trans coding, etc., as appropriate. Extensions are developed by sub classing the virtual class Extension.

The Protocol class is extended to form the AQR Protocol class. There are three capsules – Probe Capsule, Reply Capsule and AQR Data Capsule. These capsules form the protocol itself.

A) Capsule Types

A capsule is a combination of a packet and its forwarding routine; the forwarding routine is executed at every active node the capsule visits while in the network. New types of capsule, with different forwarding routines, are developed by sub classing the virtual class Capsule. The capsule starts with the sequence id of the data capsule. Then the timestamp of when the capsule is sent is stored. Then some flags and index values are stored. The path index is the pointer to the next node to reach. The path valid is a flag that is set true when the capsule carries a path to travel. Otherwise default shortest path is used to reach the destination. There is another flag, aqrFlag which is set to differentiate between capsules sent using AQR and capsules sent using SPR. Finally the path to travel is stored.

The active code of this capsule is the forwarding routine to guide the capsule to the destination. It checks for the pathValid flag and if set uses the path in the capsule to travel. Otherwise shortest path is used to forward the capsule to the destination. For valid path, the next node is obtained from the path stored using the path index pointer. The data capsule is then forwarded to this node. Once the nodes in the path stored get over, it indicates that the capsule has reached the destination. The capsule is then delivered to the receiver application.

B) Application

Applications are the entities that make use of the network to send and receive capsules as well as run independent activities. New applications are developed by sub classing the virtual class Application. It provides access to the node and its services.

C) Other modules

The data traffic is generated by a thread AQR Data Sender. It is run on the sender node. This data traffic is received by another application on the receiver node, AQR Data Receiver. There are also other utility classes.

4 Configuration

Experimenting with an active network requires that a network topology and ANTS provides some tools and infrastructure conventions to automate this process. First, entire network configurations, node addresses, applications and all initialization parameters are specified text files that are read by Configuration Manager Class to start one of its nodes locally. This includes creating the node runtime, applications, and extensions, connecting them to each other, and starting their operation.

A) Performance Analysis

The algorithm is run under various test scenarios and the test results are presented in this section. The QoS performance is analyzed for data traffic with and without background traffic .TG is a packet Traffic Generator (TG) tool that can be used to characterize the performance of packet-switched network communication protocols. The TG program generates and receives one-way packet traffic streams transmitted from the UNIX user level process between traffic source and traffic sink nodes in a network. TG is used for generating the background traffic. The TG serving as the traffic source always logs datagram transmit times. This mode of operation may be useful for analyzing network blocking characteristics or for loading a network. The TG serving as a traffic sink logs all received data grams. The behavior of the protocol is analyzed after the intermediate routers are loaded to make the default shortest path congested. For the same topology the shortest path routing is tested. Two applications are run on the source and destination nodes. The active code in the capsule is used to route the capsule through the shortest path. The delay changes and the routes taken are recorded and analyzed.

B) Performance analysis without background traffic

In this test run, the data capsules of length 100 bytes are used to test the protocol. No background traffic is used to load the shortest path. Instead the routers along the shortest path are loaded to increase the delay along the shortest path. In this scenario the test is run and the results shown below. The delay values are relative and are not exact with respect to the sender in all the data presented.

C) Performance analysis with background traffic type 1

In this test run, the data capsules of length 100 bytes are used to test the protocol. Also background traffic is introduced along the shortest path to increase the delay along the shortest path. The traffic is introduced using the tool TG. The background traffic is udp. Packets of length 500 bytes at a rate of 100Mbps are used for the traffic.

5 Fast Attack Detection using Neural Networks

Finding a certain attack, in the incoming serial data, is a searching problem. First neural networks are trained to classify attack from non attack examples and this is done in time domain. In attack detection phase, each position in the incoming matrix is tested for presence or absence of an attack. At each position in the input one dimensional matrix, each sub-matrix is multiplied by a window of weights, which has the same size as the sub-matrix. The outputs of neurons in the hidden layer are multiplied by the weights of the output layer. When the final output is high, this means that the sub-matrix under test contains an attack and vice versa. Thus, we may conclude that this searching problem is a cross correlation between the incoming serial data and the weights of neurons in the hidden layer.

The convolution theorem in mathematical analysis says that a convolution of f with h is identical to the result of the following steps: let F and H be

the results of the Fourier Transformation of f and h in the frequency domain. Multiply F and H* in the frequency domain point by point and then transform this product into the spatial domain via the inverse Fourier Transform. As a result, these cross correlations can be represented by a product in the frequency domain. Thus, by using cross correlation in the frequency domain, speed up in an order of magnitude can be achieved during the detection process [1,2,3,4,5,7,8,9,14]. Assume that the size of the attack code is 1xn. In attack detection phase, a sub matrix I of size 1xn (sliding window) is extracted from the tested matrix, which has a size of 1xN. Such sub matrix, which may be an attack code, is fed to the neural network. Let W_i be the matrix of weights between the input sub-matrix and the hidden layer. This vector has a size of 1xn and can be represented as 1xn matrix. The output of hidden neurons h(i) can be calculated as follows:

$$h_i = g\left(\sum_{k=1}^{n} W_i(k) I(k) + b_i \right) \quad (1)$$

where g is the activation function and b(i) is the bias of each hidden neuron (i). Equation 1 represents the output of each hidden neuron for a particular sub-matrix I. It can be obtained to the whole input matrix Z as follows:

$$h_i(u) = g\left(\sum_{k=-n/2}^{n/2} W_i(k)\, Z(u+k) + b_i \right) \quad (2)$$

Eq.2 represents a cross correlation operation. Given any two functions f and d, their cross correlation can be obtained by:

$$d(x) \otimes f(x) = \left(\sum_{n=-\infty}^{\infty} f(x+n) d(n) \right) \quad (3)$$

Therefore, Eq. 2 may be written as follows [1]:

$$h_i = g\left(W_i \otimes Z + b_i \right) \quad (4)$$

where h_i is the output of the hidden neuron (i) and $h_i(u)$ is the activity of the hidden unit (i) when the sliding window is located at position (u) and $(u) \in [N-n+1]$.

Now, the above cross correlation can be expressed in terms of one dimensional Fast Fourier Transform as follows [1]:

$$W_i \otimes Z = F^{-1}\left(F(Z) \bullet F^*\left(W_i \right) \right) \quad (5)$$

Hence, by evaluating this cross correlation, a speed up ratio can be obtained comparable to conventional neural networks. Also, the final output of the neural network can be evaluated as follows:

$$O(u) = g\left(\sum_{i=1}^{q} W_o(i)\, h_i(u) + b_o \right) \quad (6)$$

where q is the number of neurons in the hidden layer. O(u) is the output of the neural network when the sliding window located at the position (u) in the input matrix Z. W_o is the weight matrix between hidden and output layer.

The complexity of cross correlation in the frequency domain can be analyzed as follows:

1- For a tested matrix of 1xN elements, the 1D-FFT requires a number equal to $N\log_2 N$ of complex computation steps [13]. Also, the same number of complex computation steps is required for computing the 1D-FFT of the weight matrix at each neuron in the hidden layer.

2- At each neuron in the hidden layer, the inverse 1D-FFT is computed. Therefore, q backward and (1+q) forward transforms have to be computed. Therefore, for a given matrix under test, the total number of operations required to compute the 1D-FFT is $(2q+1)N\log_2 N$.

3- The number of computation steps required by FTDNNs is complex and must be converted into a real version. It is known that, the one dimensional Fast Fourier Transform requires $(N/2)\log_2 N$ complex multiplications and $N\log_2 N$ complex additions [13]. Every complex multiplication is realized by six real floating point operations and every complex addition is implemented by two real floating point operations. Therefore, the total number of computation steps required to obtain the 1D-FFT of a 1xN matrix is:

$$\rho = 6((N/2)\log_2 N) + 2(N\log_2 N) \quad (7)$$

which may be simplified to:

$$\rho = 5N\log_2 N \quad (8)$$

4- Both the input and the weight matrices should be dot multiplied in the frequency domain. Thus, a number of complex computation steps equal to qN should be considered. This means 6qN real operations will be added to the number of computation steps required by FTDNNs.

5- In order to perform cross correlation in the frequency domain, the weight matrix must be extended to have the same size as the input matrix. So, a number of zeros = (N-n) must be added to the weight matrix. This requires a total real number of computation steps = q(N-n) for all neurons. Moreover, after computing the FFT for the weight matrix, the conjugate of this matrix must be obtained. As a result, a real number of computation steps = qN should be added in order to obtain the conjugate of the weight matrix for all neurons. Also, a number of real computation steps equal to N is required to create butterflies complex numbers ($e^{-jk(2\Pi n/N)}$), where 0<K<L. These (N/2) complex numbers are multiplied by the elements of the input matrix or by previous complex numbers during the computation of FFT. To create a complex number requires two real floating point operations. Thus, the total number of computation steps required for FTDNNs becomes:

$$\sigma=(2q+1)(5N\log_2 N) +6qN+q(N-n)+qN+N \quad (9)$$

which can be reformulated as:

$$\sigma=(2q+1)(5N\log_2 N)+q(8N-n)+N \quad (10)$$

6- Using sliding window of size 1xn for the same matrix of 1xN pixels, q(2n-1)(N-n+1) computation steps are required when using CTDNNs for certain attack detection or processing (n) input data. The theoretical speed up factor η can be evaluated as follows:

$$\eta = \frac{q(2n-1)(N-n+1)}{(2q+1)(5N\log_2 N)+q(8N-n)+N} \quad (11)$$

CTDNNs and FTDNNs are shown in Figures 6 and 7 respectively.

Time delay neural networks accept serial input data with fixed size (n). Therefore, the number of input neurons equals to (n). Instead of treating (n) inputs, the proposed new approach is to collect all the incoming data together in a long vector (for example 100xn). Then the input data is tested by time delay neural networks as a single pattern with length L (L=100xn). Such a test is performed in the frequency domain as described in section II. The combined attack in the incoming data may have real or complex values in a form of one or two dimensional array. Complex-valued neural networks have many applications in fields dealing with complex numbers such as telecommunications, speech recognition and image processing with the Fourier Transform [6,10]. Complex-valued neural networks mean that the inputs, weights, thresholds and the activation function have complex values. In this section, formulas for the speed up ratio with different types of inputs (real /complex) will be presented. Also, the speed up ratio in case of a one and two dimensional incoming input matrix will be concluded. The operation of FTDNNs depends on computing the Fast Fourier Transform for both the input and weight matrices and obtaining the resulting two matrices. After performing dot multiplication for the resulting two matrices in the frequency domain, the Inverse Fast Fourier Transform is determined for the final matrix. Here, there is an excellent advantage with FTDNNs that should be mentioned. The Fast Fourier Transform is already dealing with complex numbers, so there is no change in the number of computation steps required for FTDNNs. Therefore, the speed up ratio in case of complex-valued time delay neural networks can be evaluated as follows:

1) In case of real inputs

A) For a one dimensional input matrix

Multiplication of (n) complex-valued weights by (n) real inputs requires (2n) real operations. This produces (n) real numbers and (n) imaginary numbers. The addition of these numbers requires (2n-2) real operations. The multiplication and addition operations are repeated (N-n+1) for all possible sub matrices in the incoming input matrix. In addition, all of these procedures are repeated at each neuron in the hidden layer. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta=2q(2n-1)(N-n+1) \quad (12)$$

The speed up ratio in this case can be computed as follows:

$$\eta = \frac{2q(2n-1)(N-n+1)}{(2q+1)(5N\log_2 N)+q(8N-n)+N} \quad (13)$$

Practical speed up ratio for searching short successive (n) data in a long input vector (L) using complex-valued time delay neural networks

is shown in Figure 8. This has beed performed by using a 700 MHz processor and MATLAB.

B) For a two dimensional input matrix

Multiplication of (n^2) complex-valued weights by (n^2) real inputs requires ($2n^2$) real operations. This produces (n^2) real numbers and (n^2) imaginary numbers. The addition of these numbers requires ($2n^2$-2) real operations. The multiplication and addition operations are repeated $(N-n+1)^2$ for all possible sub matrices in the incoming input matrix. In addition, all of these procedures are repeated at each neuron in the hidden layer. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta=2q(2n^2-1)(N-n+1)^2 \quad (14)$$

The speed up ratio in this case can be computed as follows:

$$\eta=\frac{2q(2n^2-1)(N-n+1)^2}{(2q+1)(5N^2\log_2 N^2)+q(8N^2-n^2)+N} \quad (15)$$

Practical speed up ratio for detecting (nxn) real valued submatrix in a large real valued matrix (NxN) using complex-valued time delay neural networks is shown in Fig. 9. This has beed performed by using a 700 MHz processor and MATLAB.

2) In case of complex inputs

A) For a one dimensional input matrix

Multiplication of (n) complex-valued weights by (n) complex inputs requires (6n) real operations. This produces (n) real numbers and (n) imaginary numbers. The addition of these numbers requires (2n-2) real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta=2q(4n-1)(N-n+1) \quad (16)$$

The speed up ratio in this case can be computed as follows:

$$\eta=\frac{2q(4n-1)(N-n+1)}{(2q+1)(5N\log_2 N)+q(8N-n)+N} \quad (17)$$

Practical speed up ratio for searching short complex successive (n) data in a long complex-valued input vector (L) using complex-valued time delay neural networks is shown in Fig. 10. This has beed performed by using a 700 MHz processor and MATLAB.

B) For a two dimensional input matrix

Multiplication of (n^2) complex-valued weights by (n^2) real inputs requires ($6n^2$) real operations. This produces (n^2) real numbers and (n^2) imaginary numbers. The addition of these numbers requires ($2n^2$-2) real operations. Therefore, the number of computation steps required by conventional neural networks can be calculated as:

$$\theta=2q(4n^2-1)(N-n+1)^2 \quad (18)$$

The speed up ratio in this case can be computed as follows:

$$\eta=\frac{2q(4n^2-1)(N-n+1)^2}{(2q+1)(5N^2\log_2 N^2)+q(8N^2-n^2)+N} \quad (19)$$

Practical speed up ratio for detecting (nxn) complex-valued submatrix in a large complex-valued matrix (NxN) using complex-valued neural networks is shown in Fig. 11. This has beed performed by using a 700 MHz processor and MATLAB.

An interesting point is that the memory capacity is reduced when using FTDNN. This is because the number of variables is reduced compared with CTDNN. The neural algorithm presented here can be inserted very easily in any Anti-Attack gateway software.

6. Conclusion

The AQR protocol is implemented using ANTS and the performance of the AQR algorithm is analyzed. It is inferred that with background traffic, the AQR finds alternative paths quickly. It reduces the delay and subsequently reduces the jitter involved. The variant of AQR using the product of available bit rate and delay for finding the best path is useful in reducing the jitter where the overall traffic in the network is heavy. It helps to maintain the jitter in networks with more bursty traffic. It is also inferred that the performance of AQR is better than that of SPR in both cases. Performance is analyzed for various traffic classes. The probe capsules are sent along each path from the source to the destination to find the best path. By tuning the probing frequency to an optimal value, the traffic caused by the probe

capsules can be reduced. Comparing with the data traffic, probe capsules are small in number. The topology should be known by all nodes to find the best path. In this paper dynamic topology is considered and flooding is used to find the best path. This finds application in finding effective QoS routing in ad-hoc networks as well as defending DDoS attacks by identifying the attack traffic path using QoS regulations. The flooding overhead involved is unavoidable and some pay off measures need to be identified. It is also proposed to choose the best path based on error rate so that the loss of probe capsules can also be taken into consideration. A new approach for fast attack detection has been presented. Such strategy has been realized by using a new design for time delay neural networks. Theoretical computations have shown that FTDNNs require fewer computation steps than conventional ones. This has been achieved by applying cross correlation in the frequency domain between the incoming serial data and the input weights of time delay neural networks. Simulation results have confirmed this proof by using MATLAB. Furthermore, the memory complexity has been reduced when using the fast neural algorithm. In addition, this algorithm can be combined in any Anti-attack gateway software.

References:

[1] H. M. El-Bakry, and Q. Zhao, "A Modified Cross Correlation in the Frequency Domain for Fast Pattern Detection Using Neural Networks," International Journal of Signal Processing, vol.1, no.3, pp. 188-194, 2004.

[2] H. M. El-Bakry, and Q. Zhao, "Fast Object/Face Detection Using Neural Networks and Fast Fourier Transform," International Journal of Signal Processing, vol.1, no.3, pp. 182-187, 2004.

[3] H. M. El-Bakry, and Q. Zhao, "Fast Pattern Detection Using Normalized Neural Networks and Cross Correlation in the Frequency Domain," EURASIP Journal on Applied Signal Processing, Special Issue on Advances in Intelligent Vision Systems: Methods and Applications—Part I, vol. 2005, no. 13, 1 August 2005, pp. 2054-2060.

[4] H. M. El-Bakry, and Q. Zhao, "A Fast Neural Algorithm for Serial Code Detection in a Stream of Sequential Data," International Journal of Information Technology, vol.2, no.1, pp. 71-90, 2005.

[5] H. M. El-Bakry, and H. Stoyan, "FNNs for Code Detection in Sequential Data Using Neural Networks for Communication Applications, " Proc. of the First International Conference on Cybernetics and Information Technologies, Systems and Applications: CITSA 2004, 21-25 July, 2004. Orlando, Florida, USA, Vol. IV, pp. 150-153.

[6] A. Hirose, "Complex-Valued Neural Networks Theories and Applications", Series on innovative Intellegence, vol.5. Nov. 2003.

[7] H. M. El-Bakry, "Face detection using fast neural networks and image decomposition," Neurocomputing Journal, vol. 48, 2002, pp. 1039-1046.

[8] H. M. El-Bakry, "Human Iris Detection Using Fast Cooperative Modular Neural Nets and Image Decomposition," Machine Graphics & Vision Journal (MG&V), vol. 11, no. 4, 2002, pp. 498-512.

[9] H. M. El-Bakry, "Automatic Human Face Recognition Using Modular Neural Networks," Machine Graphics & Vision Journal (MG&V), vol. 10, no. 1, 2001, pp. 47-73.

[10] S. Jankowski, A. Lozowski, M. Zurada, " Complex-valued Multistate Neural Associative Memory," IEEE Trans. on Neural Networks, vol.7, 1996, pp.1491-1496.

[11] H. M. El-Bakry, and Q. Zhao, "Fast Pattern Detection Using Neural Networks Realized in Frequency Domain," Proc. of the International Conference on Pattern Recognition and Computer Vision, The Second World Enformatika Congress WEC'05, Istanbul, Turkey, 25-27 Feb., 2005, pp. 89-92.

[12] H. M. El-Bakry, and Q. Zhao, "Sub-Image Detection Using Fast Neural Processors and Image Decomposition," Proc. of the International Conference on Pattern Recognition and Computer Vision, The Second World Enformatika Congress WEC'05, Istanbul, Turkey, 25-27 Feb., 2005, pp. 85-88.

[13] J. W. Cooley, and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput. 19, 297–301 (1965).

[14] R. Klette, and Zamperon, "Handbook of image processing operators, " John Wiley & Sonsltd, 1996.

[15] Michael Welzl, Alfred Cihal, Max Mühlhäuser, “An Approach to Flexible QoS Routing with Active Networks”, Proceedings of the Fourth Annual International Workshop on Active Middleware Services (AMS’02), 2002.

[16] Wang Jianxin, Wang Weiping, Chen Jian'er, Chen Songqiao.” A randomized QoS routing algorithm on networks with inaccurate link-state information”, Proc. the 16th World Computer Conference, International Conference of Communication Technology, Beijing, Aug., 2000, pp.1617-1622.

[17] Stavros Vrontis, Irene Sygkouna, Maria Chantzara and Eystathios Sykas, “Enabling Distributed QoS Management utilizing Active Network technology”, National Technical University of Athens.

[18] David J. Wetherall, John V. Guttag and David L. Tennenhouse, “ANTS: A Toolkit for Building and Dynamically Deploying Networks Protocols”, Software Devices and Systems Group, Laboratory for Computer Science, Massachusetts Institute of Technology, April 1998.

[19] Juraj Sucík, Ing. František Jakab, “Measurement and Evaluation of Quality of Service Parameters in Computer Networks”, Department of Computers and Informatics, Technical University of Košice, Letná 9, 041 20 Košice, Slovak Republic.

[20] Roche A. Guérin , Ariel Orda, “QoS routing in networks with inaccurate information: Theory and algorithms”, IEEE/ACM Transactions on Networking (TON), v.7 n.3, p.350-364, June 1999

[21] Dean H. Lorenz , Ariel Orda, “QoS routing in networks with uncertain parameters”, IEEE/ACM Transactions on Networking (TON), v.6 n.6, p.768-778, Dec. 1998

[22] Rajagopalan B, Saadick H, “A Framework for QoS-based Routing in the Internet”, RFC 2386, IETF, Aug., 1998.

[23] M. Reisslein, K. W. Ross, and S. Rajagopal, ”Guaranteeing statistical QoS to regulated traffic: The single node case”, Proceedings of IEEE INFOCOM’99, pages 1061–1062, New York, March 1999.

[24] M. Reisslein, K. W. Ross, and S. Rajagopal. Guaranteeing statistical QoS to regulated traffic: The multiple node case”, Proceedings of 37th IEEE Conference on Decision and Control (CDC), pages 531–531, Tampa, December 1998.

[25] David Wetherall, Ulana Legedza, and John Guttag, “Introducing New Internet Services: Why and How”, Software Devices and Systems Group, Laboratory for Computer Science, Massachusetts Institute of Technology, July 1998.

[26] P. Hurley, J.-Y. Le Boudec, P. Thiran, and M. Kara.ABE,” Providing a Low-Delay Service within Best Effort”, IEEE Network Magazine, v 15, no 3,May/June 2001.

Table 1. Performance data without background traffic.

	SPR	AQR
Max. delay (ms)	8364	7006
Min. delay (ms)	5554	5554
Avg. delay (ms)	5760	5584
Max. jitter (ms)	2810	1452

Table 2. Performance data with background traffic type 1

	SPR	AQR
Max. delay (ms)	1825	1827
Min. delay (ms)	1806	1806
Avg. delay (ms)	1808	1806
Max. jitter (ms)	19	21

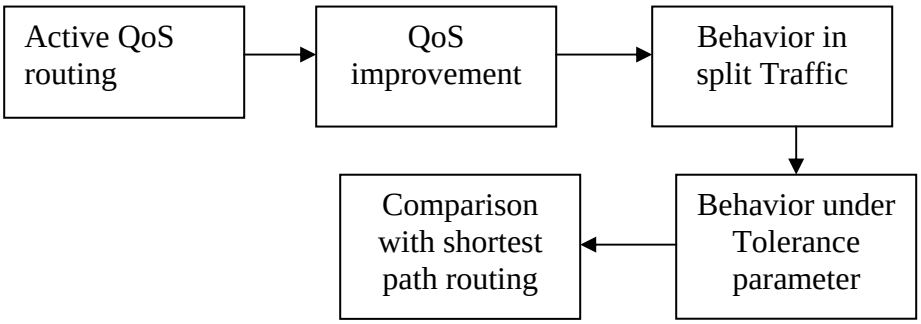

Fig. 1. Data flow diagram.

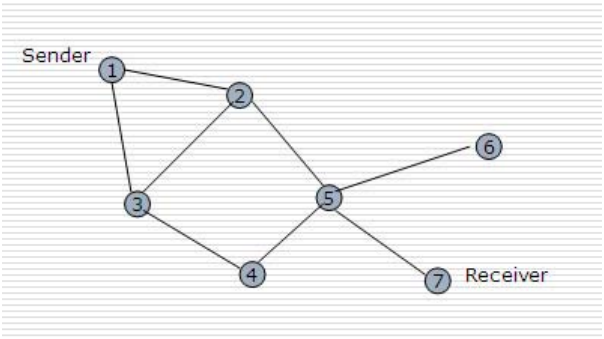

Fig. 2. Sample network topology.

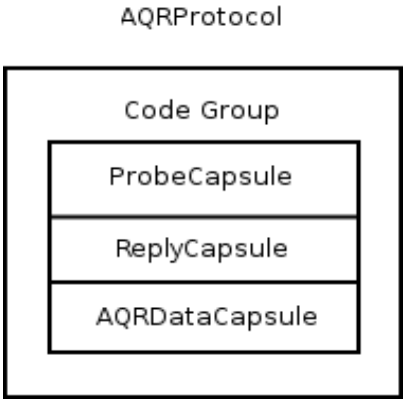

Fig. 3. Structure of AQR protocol.

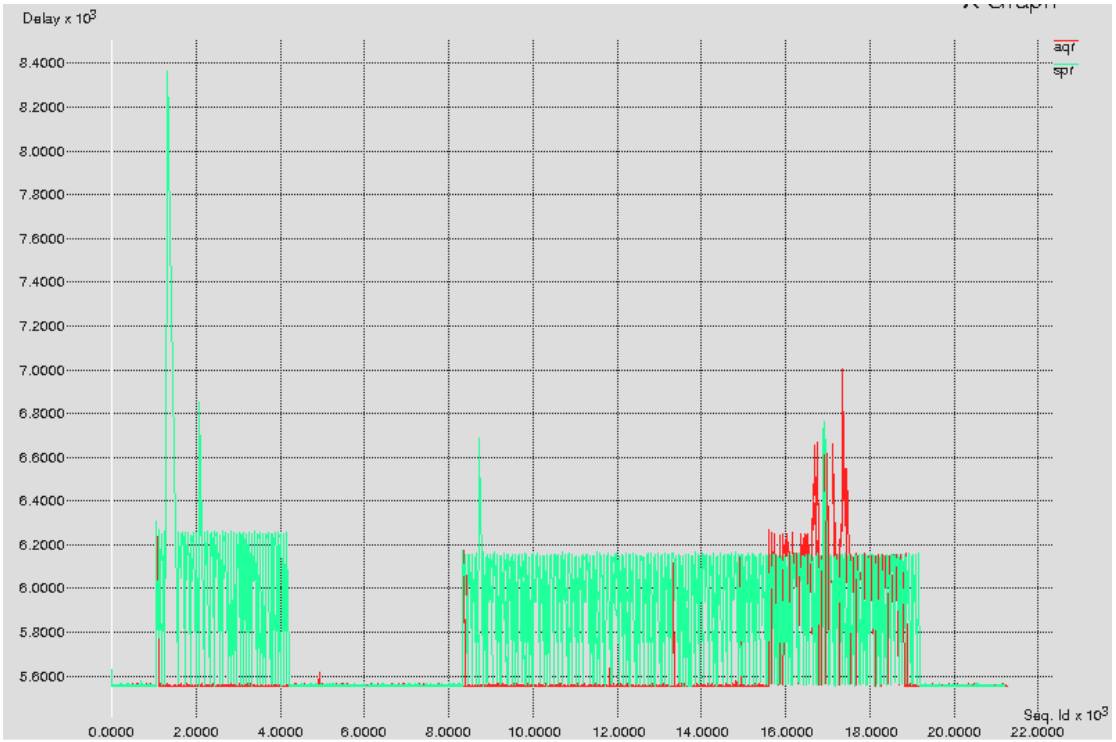

Fig. 4 Delay response without background traffic.

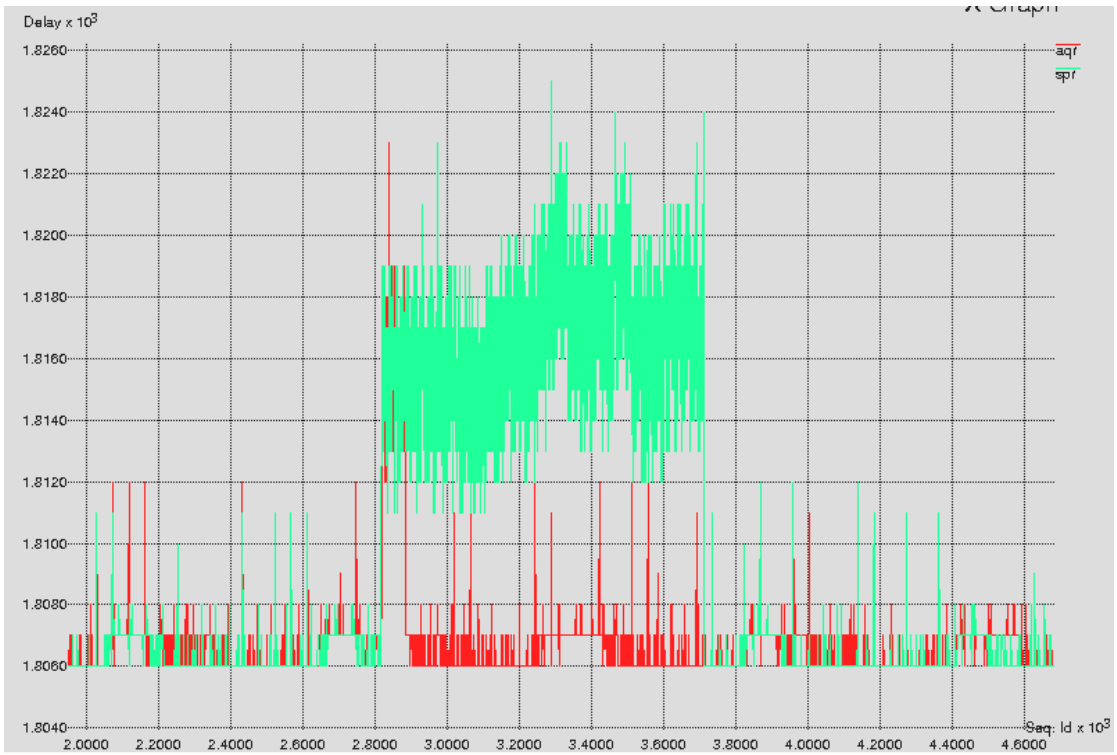

Fig.5. Delay response with background traffic type 1.

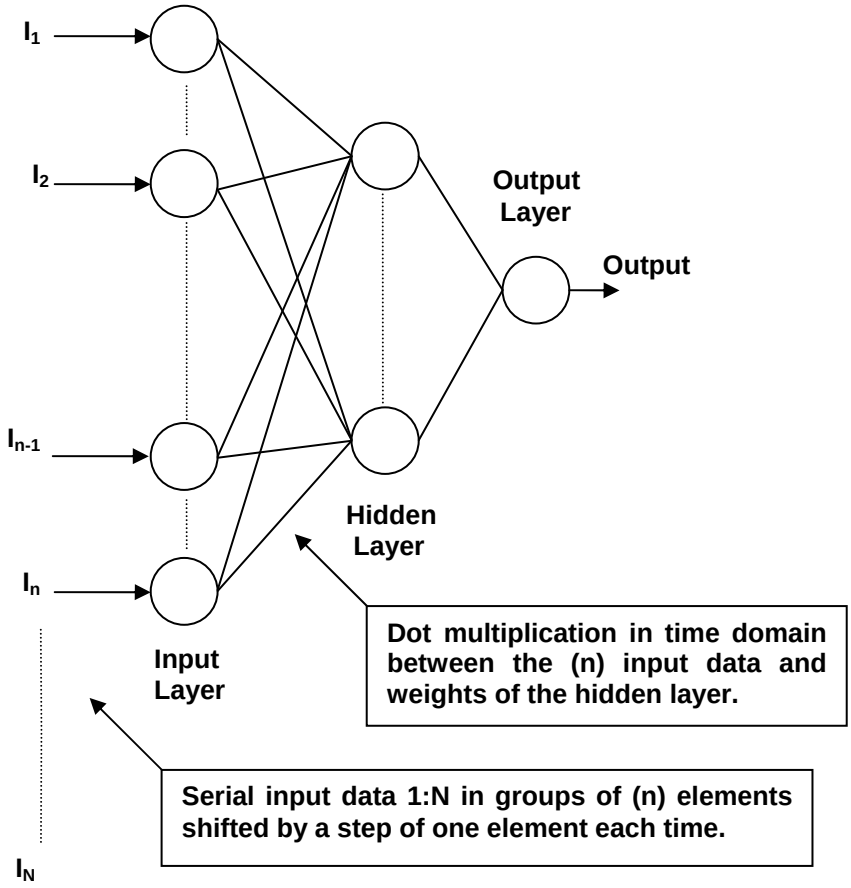

Fig.6. Classical time delay neural networks.

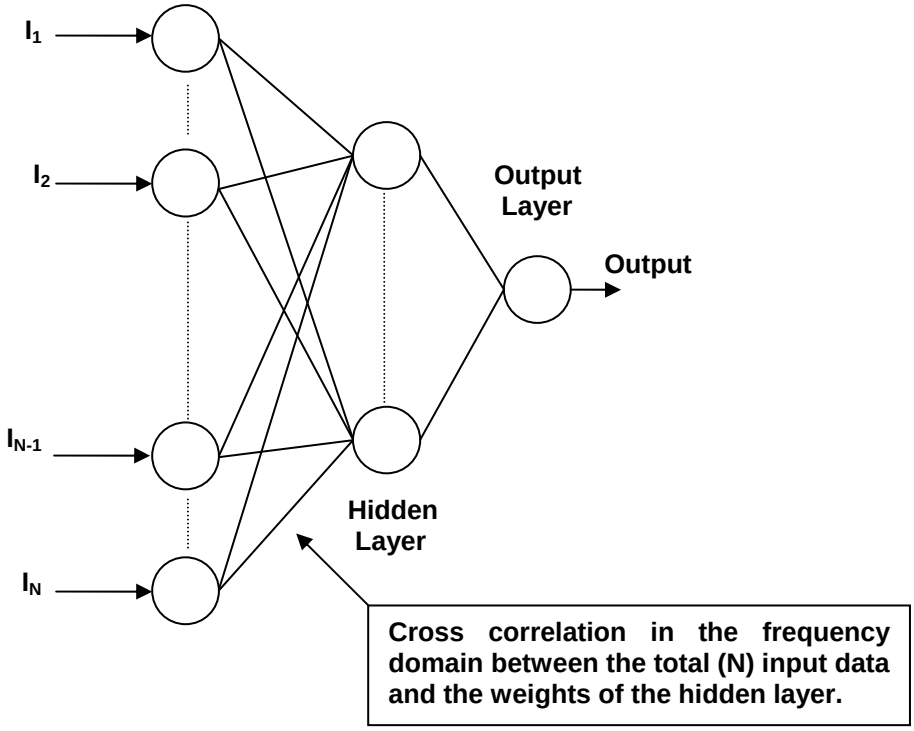

Fig.7. Fast time delay neural networks.

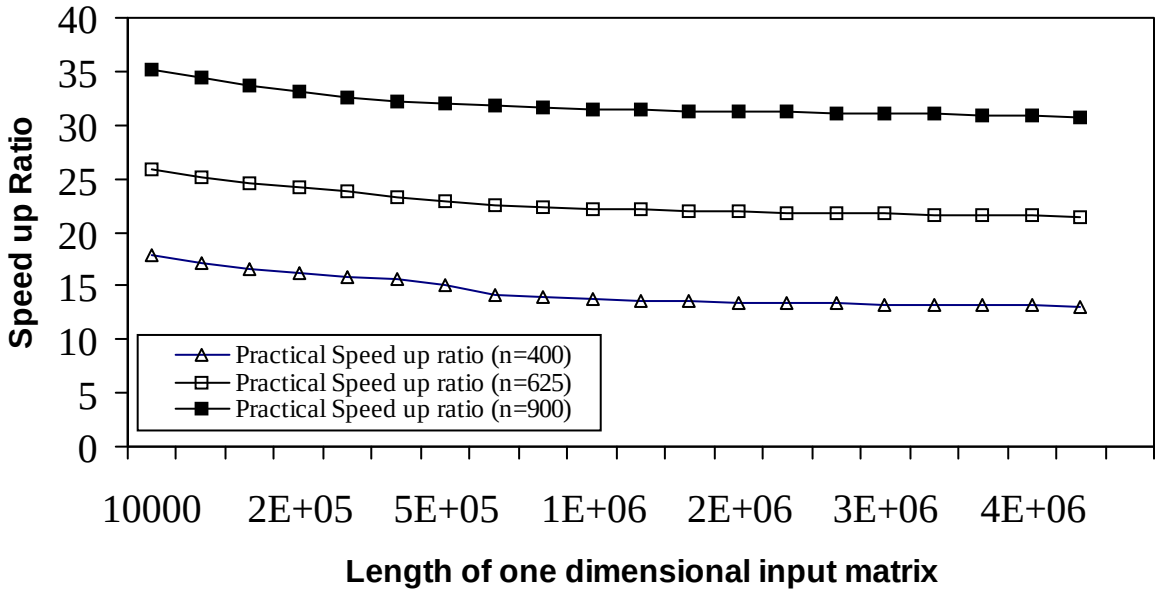

Fig. 8. Practical speed up ratio for time delay neural networks in case of one dimensional real-valued input matrix and complex-valued weights.

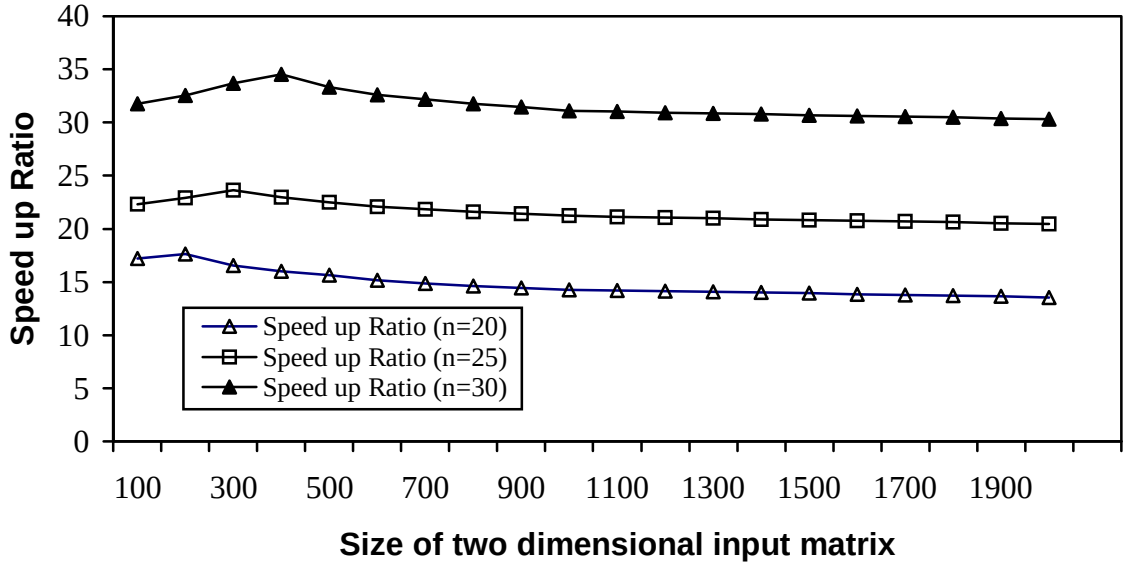

Fig. 9. Practical speed up ratio when using FTDNNs in case of two dimensional real-valued input matrix and complex-valued weights.

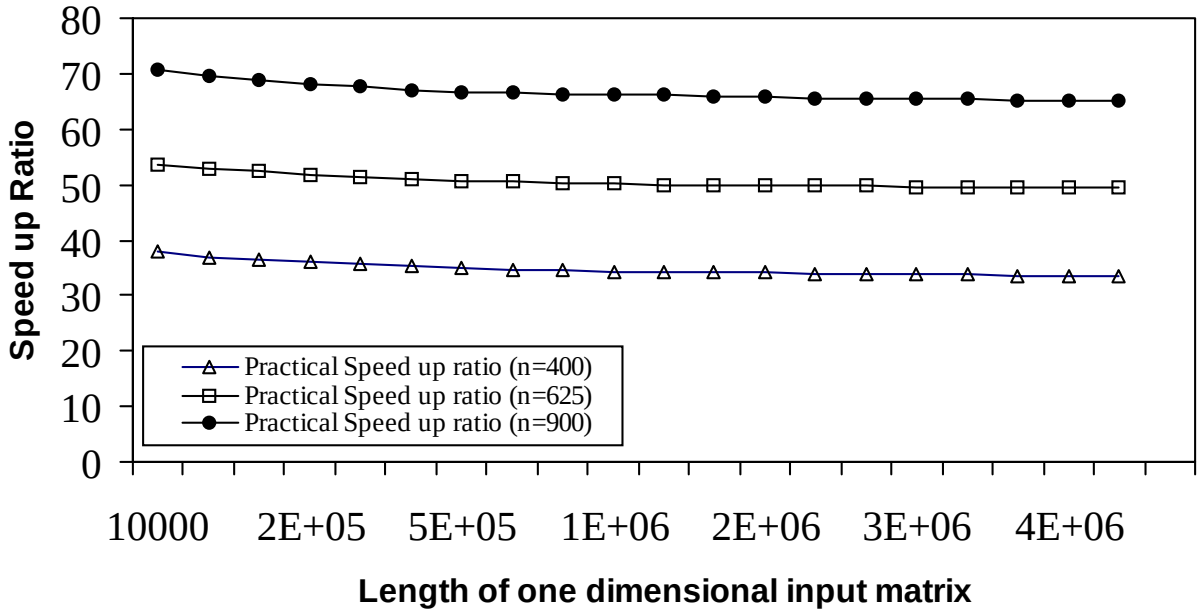

Fig. 10. Practical speed up ratio when using FTDNNs in case of one dimensional complex-valued input matrix and complex-valued weights.

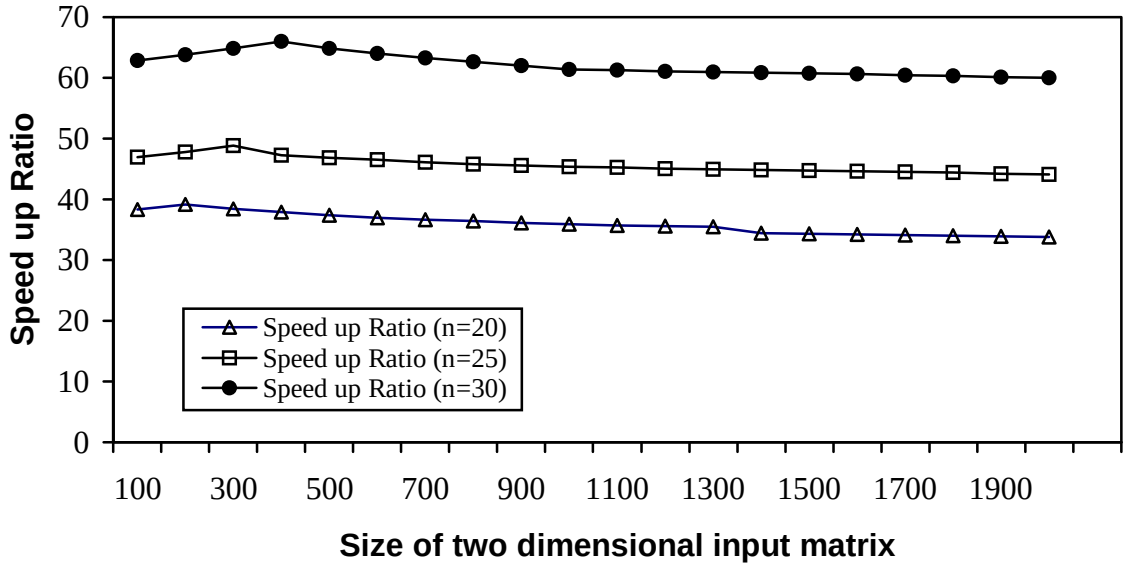

Fig. 11. Practical speed up ratio when using FTDNNs in case of two dimensional complex-valued input matrix and complex-valued weights.