An Innovative Approach for Online Bandwidth Adjustment in IP DiffServ Networks Using Learning Automata

Mohsen Jahanshahi
Department of Computer Engineering
Islamic Azad University – Central Tehran Branch
Tehran, Iran
Mjahanshahi@iauctb.ac.ir

Mohammad Reza Meybodi
Department of Computer Engineering
Amirkabir University of Technology
Tehran, Iran
Mmeybodi@aut.ac.ir

Abstract - Audio and video conferencing are two examples of the applications that need a network for providing QoS guarantee according to the available bandwidth. A very important factor in QoS is end-to-end delay. The destination node receives the packets after they go through the routers queue. Therefore, if the network guarantees the queuing delay in routers, the network will find the ability to guarantee end-to-end delay. Moreover, the developers too will find the opportunity to sign contracts which are based on Service Level Agreement (SLA). Queuing delay can be solved with using bandwidth provisioning. Typically there are many ways for band provisioning. But just two methods concentrate on the queuing delay in router queue. However due to some shortcomings, these methods are not suitable for our aim. What we have presented in this paper is method which guarantees queuing delay of three service classes of DiffServ networks at the same time. The method that we have suggested uses learning automata, and it will not be necessary to have any knowledge or presumptions of traffic model and network parameter for using it. Thus one can use our adaptive method online on a network which has various features without reconfiguration. On the other hand the computational complexity of our method is very low and this feature enables it to be applied on high-speed networks. We have also evaluated the efficiency of our method in comparison to the previous methods.

Index Terms - Bandwidth provisioning, DiffServ architecture, Learning automata, Delay guarantee

I. INTRODUCTION

Two architectures named integrated service and differentiated service for quality of-service (QoS) are presented by IETF. Differentiated service (DiffServ) architecture is designed to cope with the drawbacks of integrated service. DiffServ is designed so that a different behavior can be utilized for packets of different class. This architecture supposes that packets which belong to same class, regardless of their flows, have the same QoS requirements [8]. This architecture provides three classes of service; 1) EF (Expedited Forwarding) which is very sensitive to packet loss rate, delay, and jitter. This class is named also premium service. 2) AF (Assured Forwarding) which is proper for network management protocols and 3) BE (Best Effort) in which maximum effort is done to provide service for the packets of this class, but in critical situations the packets of this service class maybe misbehaved with[8, 9]. In this architecture there is a classifier which classifies traffic to the different classes.

Some applications such as audio and video conferencing require a network to provide QoS guarantee with respect to available bandwidth. End-2-End delay is one of the prominent factors in QoS. Packets after crossing the routers queue arrive to destination node. Thus with guaranteeing queuing delay in routers the network will be able to guarantee End-2-End delay. Bandwidth provisioning is one of the methods for guaranteeing the queuing delay. Based on controlling methods which they use, adaptive bandwidth control (ABC) algorithms are classified to two categories; closed-loop or open-loop [1]. In closed-loop methods some metrics such as the packet loss [2-4], average queue length [5], delay [6-7], or other system states are checked as feedback to adjust new bandwidth. Generally, there are just two methods which have worked on queuing delay. These methods along with their drawbacks are stated as follows; in [7] initially a presumption of queue length distribution is considered. Then statistic of queue length is compared with this presumption and then related modifications are performed. Drawbacks of this method are as follows; this method requires a presumption distribution of queue length. Since the traffic in not predictable, this distribution can not be specified properly. Therefore, this is subject to error. Furthermore, this method is designed for a single queue. Also, in this method, only exponential distribution in order to approximate the new service rate is used. But approximation of this value for other distributions is very difficult [1]. Similarly, in this reference probability of that queuing delays is more than maximum acceptable delay is less than $\varepsilon$. That means delay is not guaranteed with probability of one. In [6] authors try to keep both packet loss rate and queuing delay in a desirable level. Note that this method doesn’t guarantee the queuing delay. But its object is providing a tradeoff between packet loss rate and queuing delay. In this method input curve is compared with output curve and then necessary modification is performed. This method has also some drawbacks as follows; this method has high computational complexity. Therefore, it is not proper for high speed networks. Furthermore, this method has not scalability. The authors of this reference show that this method can not keep both of the parameters in a desirable level in special situations.

Bandwidth adjustment in DiffServ networks involves the determination of bandwidth amount that will be allocated for each class of DiffServ network. In [17-20], some approaches
for dynamic bandwidth provisioning are proposed which utilize the artificial intelligence methodologies presented in [21-30]. These researches are based on TD-learning methods. In this paper, using learning automata, an innovative method for guaranteeing simultaneously queuing delay of three service class of DiffServ network is proposed. Structure of this paper is as follows; in section 2, learning automata is introduced. In section 3, proposed method is presented. Simulation model is presented in section 4. In section 5, simulation result is presented. Also, efficiency of the proposed method, in comparison to previous methods, is evaluated. Conclusion of this research comes in section 7.

II. LEARNING AUTOMATA

Learning automata is an abstract model that chooses an action from a finite set of its actions randomly and takes it. In this case, environment evaluates this taken action and responses by a reinforcement signal. Then, learning automata updates its internal information regarding both the taken action and received reinforcement signal. After that, learning automata chooses another action again. Figure 1 depicts the relationship between learning automata and environment. Every environment is represented by \( E = \{\alpha, \beta, c\} \), where \( \alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_s\} \) is a set of inputs, \( \beta = \{\beta_1, \beta_2, \ldots, \beta_r\} \) is a set of outputs, and \( c = \{c_1, c_2, \ldots, c_i\} \) is a set of penalty probabilities. Whenever set \( \beta \) has just two members, model of environment is \( p \) model. In this environment \( \beta_1 = 1 \), \( \beta_2 = 0 \) are considered as penalty and reward respectively. Similarly, \( Q \) model of environment contains a finite set of members. Also, \( S \) model of environment has infinite number of members. \( c_i \) is the penalty probability of taken action \( \alpha_i \). Learning automata is classified into fixed structure and variable structure. Learning automata with variable structure is introduced as follows; Learning automata with variable structure is represented by \( \{\alpha, \beta, p, T\} \), where \( \alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_s\} \) is a set of actions, \( \beta = \{\beta_1, \beta_2, \ldots, \beta_r\} \) is a set of inputs, \( p = \{p_1, p_2, \ldots, p_r\} \) is the action probability vector, and \( p(n+1) = T[\alpha(n), \beta(n), p(n)] \) is learning algorithm. Learning automata operates as follows; learning automata chooses an action from its probability vector randomly \( (P_i) \) and takes it. Suppose that the chosen action is \( \alpha_i \). Learning automata after receiving reinforcement signal from environment updates its action probability vector according to formulas 1 and 2 in case of desirable and undesirable received signals respectively. In formulas 1 and 2, \( a \) and \( b \) are reward and penalty parameters respectively.

\[
\begin{align*}
p_j(n+1) &= p_j(n) + a(1-p_j(n)) \quad \forall j \in \{1, 2, \ldots, r\} \quad (1) \\
p_j(n+1) &= p_j(n) - a_j p_j(n) \quad \forall j \notin \{i\} \\
p_i(n+1) &= (1-b_j) p_i(n) \quad \forall j \notin \{i\} \\
p_j(n+1) &= \frac{b_j}{r-1}(1-b_j) p_j(n) \quad \forall j \notin \{i\} \quad (2)
\end{align*}
\]

III. PROPOSED METHOD

The objective of this research is presenting a method based on learning automata for guaranteeing simultaneously queuing delay of 2ms and 3ms for EF and AF service class of DiffServ networks respectively. Type of used automata in this research is stochastic automata with variable structure. Also, the action probability vector is updated according to \( L_{R-P} \) algorithm. In proposed method, learning automata in order to perceive the state of environment uses a function which is presented in figure 2. This function gets the queuing delay of each service class as input and then returns a reinforcement signal \( \beta \) as output. In this function a variable \( P \) for each queue is allocated \((P_0, P_1, \text{and} P_2)\). These variables are binary i.e. they can be set only to one or zero. This function operates as follows; if each queuing delay is less than its maximum acceptable delay, or the difference between current queuing delay and maximum acceptable delay is less than difference between previous queuing delay and maximum acceptable delay, then the variable \( p \) related to this queue will be set to zero, otherwise set to one. Then these three variables of \( p(\text{one} p \text{for each queue of service class}) \), which are binary, are mapped together to a number in the range of 0 to 7 as a state of environment. If the new state of environment is less than the state of before taking the action, then the reinforcement signal \( \beta \) is set to zero, otherwise it is set to one. Learning automata actions and effect of each action are presented in table I. As it is seen in table I, learning automata has nine valid actions.

<table>
<thead>
<tr>
<th>Table I</th>
<th>LEARNING AUTOMATA ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action description</td>
<td>Action number</td>
</tr>
<tr>
<td>Decrease 3 units from service rate of EF queue</td>
<td>0</td>
</tr>
<tr>
<td>Increase 3 units from service rate of EF queue</td>
<td>1</td>
</tr>
<tr>
<td>Decrease 3 units from service rate of AF queue</td>
<td>2</td>
</tr>
<tr>
<td>Increase 3 units from service rate of AF queue</td>
<td>3</td>
</tr>
<tr>
<td>Decrease 3 units from service rate of BE queue</td>
<td>4</td>
</tr>
<tr>
<td>Increase 3 units service rate of BE queue</td>
<td>5</td>
</tr>
<tr>
<td>No change in service rate of EF queue</td>
<td>6</td>
</tr>
<tr>
<td>No change in service rate of AF queue</td>
<td>7</td>
</tr>
<tr>
<td>No change in service rate of BE queue</td>
<td>8</td>
</tr>
</tbody>
</table>
Algorithm of the proposed method which is run every two seconds by router is as follows; initially learning automata chooses one of its nine actions from action probability vector. This action is performed by the router. After two seconds, queuing delay for each service class is computed simultaneously. These three queuing delays, as seen in figure 2, together are mapped to a state of environment by a function. If the new state of environment is less than state of before taking the action, the reinforcement signal $\beta$ is set to zero, otherwise it is set to one. Considering the value of $\beta$, action probability vector is updated according to $L_{R-P}$ algorithm. Learning automata, after updating this vector, chooses another action again.

**IV. SIMULATION MODEL**

In order to evaluate the proposed method, the network model which is designed in OpNet simulator is used (figure 3). As seen in figure 3, there is a classifier which classifies packets to the different classes based on their TOS field. Since Poisson traffic generation model is not proper for generating the burst traffic [14], in this model the Pareto traffic generation model is used in order to generate the burst traffic (self-similar). Sources of traffic generate EF, AF and BE traffics randomly. Also in this model value of parameter HURST which specifies the intensity of burst traffic is set to 0.7.

The capacity of router’s output link is 30 packets per second. Task of learning automata is allocation of this capacity among three queues of each service class so that the desirable queuing delay is guaranteed for each service class simultaneously. Ince learning automata can choose an action among nine valid actions, initially the probability of all these actions is equal and set to \(\frac{1}{9}\).
V. SIMULATION RESULTS

The objective of this research is to guarantee simultaneously queuing delay of 2ms and 3ms for queues of EF and AF service class respectively. Result shown in figures 4 and 5 are obtained from one simulation with learning parameter $\alpha = 0.1$. In figures 4 and 5 three charts are presented together. That means these results are obtained simultaneously and in parallel. Figure (4-a) is related to service rate of three service class queue simultaneously. Similarly, Figure (4-b) is related to queuing delay of three service class simultaneously. Jitter is another prominent factor of QoS. Some applications such as audio and video conferencing need a network to guarantee End-2-End jitter. Packets after passing the routers queue arrive to destination node. If in routers queue jitter is decreased then End-2-End jitter is decreased as well. Figure 5 shows the queuing jitter for three service classes which are obtained simultaneously in this simulation. Average of queuing jitter for EF and AF classes are 0.00102 and 0.00115 respectively. Table II presents the impact of $\alpha$ variations on three queuing delays simultaneously. Word ‘YES’ means that the proposed method can guarantee queuing delay for three service classes simultaneously. Also word ‘NO’ means the proposed method can’t guarantee queuing delay for EF or AF or both. As seen in this table learning automata, in wide range of values for learning parameter $\alpha$, can guarantee the desirable queuing delay for three service class simultaneously by dynamic sharing of output link capacity.
VI. Evaluation of the Proposed Method in Comparison to Methods [17-20]

In all methods we increased number of actions of all agents in order to improve their adaption time. But did not obtain any better result Methods which are presented in [17-20] utilize the proposed artificial intelligence methodologies presented in [21-30]. In this section the proposed method is compared with the methods presented in [17-20].

- **With respect to need for memory requirement:**
The proposed method needs \( \# \) (value of number of actions) memory space for running the algorithm which is minimal among the TD-learning-based methods. In learning-automata-based method, the actions probability vector is just sustained. Memory requirements for TD-learning-based methods are shown in table III. Therefore, the proposed method regarding memory requirements is preferred to TD-learning-based methods.

- **With respect to queuing jitter:**
Queuing jitter for both EF and AF queues in the proposed method as seen in table III among the TD-learning-based methods is minimal. As seen in figure 5 closing almost minute one the jitter is constant. Thus, it is ideal for audio and video applications.

- **With respect to computational complexity:**
This comparison is depicted in table III. Number 1 means the lower computational complexity. As seen in this table the proposed method in comparison to TD-learning-based methods has lower computational complexity and hence has less time ordering.

- **With respect to number of learning parameters:**
While number of learning parameters in TD-learning methods is four, number of learning parameters in proposed method is just one (is named learning rate) which is minimal. Since there are less learning parameters in proposed method, the parameter tuning in comparison to TD-learning-based methods is done more simply and rapidly.

- **With respect to adaption time:**
Adaption time means after how long the method can get three delays under desirable levels of three classes simultaneously and keep them in this level. The method based on learning automata with respect to speed, next to the method based on hierarchical q-learning, stands on the second place.

Table III and IV summarize the comparison among proposed method and the methods based on TD-learning.

VII. Conclusion

In this paper initially learning automata is introduced. Then a new method using learning automata for bandwidth provisioning in DiffServ networks is presented. The proposed method does not require to any knowledge and presumptions of traffic model and network parameters. This feature leads to apply the proposed method online on a network with different characteristics without any reconfiguration. That means the proposed method is adaptive. Computational complexity in the proposed method is low. Therefore it can be applied on high speed networks. Examination results demonstrate that the proposed method with respect to memory space, jitter, computational complexity, and number of learning parameters in comparison to TD-learning-based methods is optimal. Also, results show that the proposed method, in order to achieve its goal, is not much sensitive to exactly tuning of learning parameters. Also, adaption time in the proposed method next to hierarchical q-learning is the highest among the other TD-learning-based methods. The proposed method can be used in real-time networks. Since memory requirement of learning automata is minimal it can be used in sensor networks as well.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Adaption time (min/sec)</th>
<th>Required memory space for algorithm</th>
<th>Computational complexity , number and tuning the learning parameters</th>
<th>AF Queuing jitter</th>
<th>EF Queuing jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sarsa</td>
<td>3/0</td>
<td>( \text{State} # \cdot \text{Action} # )</td>
<td>2</td>
<td>0.03907</td>
<td>2.02855</td>
</tr>
<tr>
<td>Sarsa with eligibility traces</td>
<td>3/4</td>
<td>( \text{State} # \cdot \text{Action} # )</td>
<td>4</td>
<td>0.014810</td>
<td>0.05908</td>
</tr>
<tr>
<td>Q-learning</td>
<td>2/26</td>
<td>( \text{State} # \cdot \text{Action} # )</td>
<td>2</td>
<td>0.011752</td>
<td>1.15081</td>
</tr>
<tr>
<td>Q-learning with eligibility traces</td>
<td>2/30</td>
<td>( \text{State} # \cdot \text{Action} # )</td>
<td>4</td>
<td>0.03107</td>
<td>0.03675</td>
</tr>
<tr>
<td>Hierarchical q-learning</td>
<td>0/4</td>
<td>( \text{State} # \cdot \text{Agent} # \cdot \text{State} # \cdot \text{Action} # \cdot \text{Agent} # )</td>
<td>3</td>
<td>0.07447</td>
<td>0.14681</td>
</tr>
<tr>
<td>Learning automata</td>
<td>0/6</td>
<td>( \text{Action} # )</td>
<td>1</td>
<td>0.00115</td>
<td>0.00102</td>
</tr>
</tbody>
</table>
REFERENCES