

Decentralized Learning for Pricing a RED Buffer

Patrick Maillé, Bruno Tuffin, Yiping Xing, Rajarathnam Chandramouli

▶ **To cite this version:**

Patrick Maillé, Bruno Tuffin, Yiping Xing, Rajarathnam Chandramouli. Decentralized Learning for Pricing a RED Buffer. ICCCN'07 - 16th International Conference on Computer Communications and NetworkS, Aug 2007, Hawaii, United States. pp.346 - 351. hal-02165254

HAL Id: hal-02165254
https://hal.science/hal-02165254

Submitted on 25 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Decentralized Learning for Pricing a RED Buffer

Patrick Maillé
GET/ENST-Bretagne
2, rue de la Châtaigneraie CS 17607
35576 Cesson-Sévigné Cedex, FRANCE

Bruno Tuffin
IRISA/INRIA
Campus Universitaire de Beaulieu
35042 Rennes Cedex, FRANCE

Yiping Xing
and Rajarathnam Chandramouli
Stevens Institute of Technology
Hoboken, NJ 07030, USA

***Abstract*—We study a buffer that implements the Random Early Detect/Discard (RED) mechanism to cope with congestion, and offers service differentiation by proposing a finite number of slopes at different prices for the RED probability. As a characteristic, the smaller the slope, the better the resulting QoS. Users are sensitive to their average throughput and to the price they pay. Since the study of the noncooperative game played is rendered difficult by the discrete nature of the strategy sets, and since it is not likely that users have a perfect knowledge of the game but only know their experienced utility, we introduce a decentralized learning algorithm to progressively reach a Nash equilibrium over time. We examine the effect of prices on the final game outcomes.**

I. Introduction

Random Early Discard/Detection (RED) is an active queue management technique advocated for telecommunication flows, when the goal is to avoid congestion, by controlling the average queue length at a router through rate adaptation. The basic idea behind RED is simple. It drops packets randomly, with dropping probability proportional to the buffer/queue length if the length is between a certain minimum and maximum threshold values [1]. Packets are dropped with probability 1 if the length exceeds the maximum threshold. For flows conforming to the Transmission Control Protocol (TCP), packet drops indicate to the source that its rate has to be reduced. As a consequence, TCP is a good candidate for the application of RED: when this mechanism is applied to several sources in the network, the overall input rate is controlled, thereby avoiding congestion. Notice that for real-time flows sent via the User Datagram Protocol (UDP), packet losses are not taken into account by the source since the sending rate of each user is fixed, despite the fact that more packets are dropped in case of higher congestion.

With respect to the traditional *tail drop* mechanism that drops only the packets arriving when the buffer is full, RED offers two advantages. First, the randomness introduced avoids situations where packet losses occur exactly at the same time, i.e. when the buffer is full, for several sources. The TCP sources experiencing a loss consequently reduce their sending rate, and the network may become underutilized for a while, then flooded again periodically. This problem is known as *global synchronization* [1], and is solved by RED since the continuous increase in packet loss probability spreads losses over time as the buffer fills in. The second issue addressed by RED is related to fairness: RED is considered more fair than the tail drop scheme, as the flows with the largest sending rates are more likely to experience packet losses.

RED is therefore a tool for congestion avoidance but not for service differentiation. The idea of extending it to include service differentiation has already been proposed in the literature. For instance RIO deploys several drop probability functions depending on the service class (see for instance [2]). However, a disadvantage is that a there is (at least directly) no incentive for users to select classes providing lower quality of service (QoS), meaning that everyone will choose the one with the highest level, resulting in the classical RED. Weighted RED (WRED) [3] generally drops packets selectively based on IP precedence, so that packets with a higher IP precedence are less likely to be dropped than packets with a lower precedence. Thus, higher priority traffic is delivered with a higher probability than lower priority traffic. Again, no incentive is directly associated to enforce people to some classes. In [4], the authors have made an alternative proposition where there is a mapping between the RED function used (more exactly the slope in the linear part of the loss probability) and the declared willingness to pay of users: the higher the price, the lower the slope, i.e. the less likely the packets will be dropped. Here again, the scheme has a drawback: it requires that the declared price, a real number in the aforementioned model, is carried by each packet when going through the router implementing RED. From a practical side, it means that a non negligible part of the packet has to be devoted to this price information in place of data to transmit. This implementation issue leads to a signalling burden.

In this paper, we consider the same kind of model as in [4], offering service differentiation by proposing different slopes – at different prices – for the drop probability in the congestion avoidance mode. The main difference is that here a *finite* number of slopes are proposed to users instead of a free choice, so that the signalling issue can be addressed by using a very small number of bits in each packet, i.e., a very small overhead. *Pricing* is again used so that not all users do choose the best slope, and we study the user behaviour within the framework of noncooperative *game theory* [5], where the strategy of each flow/user is the choice of a RED slope, and its utility depends on the experienced

throughput and the price paid. Due to the discrete nature of the game now, the analysis is much trickier than in [4]. We then suggest that users learn how to make their decision in a decentralized manner, following the principles developped in [6], [7], [8], so that a Nash equilibrium of the game is reached. The two cases, when the flows using the buffer are TCP flows and when they are UDP flows are considered. The case when a mix of those types of flows share the same RED buffer is left for future work.

Note that pricing telecommunication networks has recently been the subject of an extensive literature (see [9], [10] for surveys) and that to our knowledge, this paper is besides [4] the only attempt to tackle the problem through RED.

The paper is organized as follows. Section II presents the basic model taken from [4] and discusses results previously obtained for a continuous strategy set. The new discretized version of the game is then presented in Section III. The decentralized learning mechanism for the discrete game is described in Section IV, and simulation results are provided in Section V, enabling to illustrate the behavior of the algorithm and to discuss the influence of prices on the game outcomes. Some conclusions and directions for future work are given in Section VI.

II. Model and previous results

A very basic model of the RED algorithm that we will use throughout the paper is as follows. Consider a buffer (a router) represented by a (virtual) *fluid* queue with service rate μ. The average virtual queue length q is compared to two thresholds $q_{\min}$ and $q_{\max}$, with $q_{\min} < q_{\max}$, in order to decide whether or not the incoming packet should be dropped. The drop probability is then as illustrated in Figure 1: it equals 0 if $q \leq q_{\min}$, 1 if $q \geq q_{\max}$, and $p_{\max}(q-q_{\min})/(q_{\max}-q_{\min})$ if $q_{\min} < q < q_{\max}$; the latter being called the congestion avoidance mode of operation. The parameter $p_{\max}$ denotes the value of the drop probability

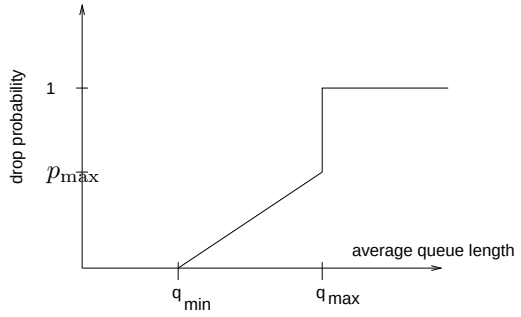


Fig. 1. Drop probability in RED as a function of q.

as the average queue length tends to $q_{\max}$ (from the left). In a best-effort network, the value of $p_{\max}$ is the same for all flows sharing the buffer. A simple principle to bring service differentiation to the network is to use different values of $p_{\max}$ (or equivalently of the slope t in the linear region) according to the service class a user belongs to. The idea is therefore to provide a smaller value of $p_{\max}$ (i.e., t) to users who pay more than the others so that, on average, they will experience less losses.

We denote the set of users (players) by $i \in \mathcal{I} = \{1, \cdots, I\}$. Let t_i be the strategy chosen by user i (here, the slope of the RED mechanism). We denote by $(t_i, [\mathbf{t}]_{-i})$ the strategy profile where user i plays t_i and all other flows $j \neq i$ use t_j from vector $[\mathbf{t}]_{-i}$. The utility of a player i depends on the whole strategy profile, which determines the average throughput of each user i and the price user i will pay. We denote by $U_i(t_i, [\mathbf{t}]_{-i})$ this utility, that user i will try to maximize.

In [4], such a model was considered, where each user could operate for a *continuous* value of the *slope* t_i in the linear region between $q_{\min}$ and $q_{\max}$. For a TCP flow i, the resulting transmission rate λ_i is then given by

$$\lambda_i = \frac{1}{R_i}\sqrt{\frac{\alpha}{p_i}}, \tag{1}$$

where R_i and p_i are TCP flow i's round trip time and drop probability, respectively. α is typically taken as $3/4$ (when delayed ack option is enabled, i.e. an acknowledgement packet is sent by the TCP receiver every two incoming packets) or $3/2$ (when it is disabled) [11]. For UDP flows, the rate λ_i is uncontrolled. We assume in this paper that all UDP flows have the same sending rate (corresponding to a given service), that we denote by λ.

In general, since the bottleneck queue is seen as a fluid queue, we can write that at equilibrium

$$\sum_{j\in\mathcal{I}} \lambda_j(1-p_j) = \mu.$$

This leads to a system of $I+1$ equations with $I+1$ unknowns. The average throughput that each user experiences as a function of the strategy profile has been determined in [4]. We quote here those results, that will be used in the remaining of the paper.

Case of only UDP users: In this case again, the sources do not take into account packet losses, so the sending rate λ for each user is fixed.

Proposition 1 ([4]): If all flows are UDP flows and the RED slope of each user $i \in \mathcal{I}$ is t_i, then the packet loss probability p_j for a player j is

- if $n \times \lambda \leq \mu$ (no congestion), then $p_j = 0$,
- else
$$p_j = t_j \frac{n\lambda - \mu}{\lambda \times \sum_k t_k}.$$

Case of only TCP users: TCP flows take into account packet losses: they are considered as congestion signals that indicate the source to reduce its sending rate. Therefore the interaction between TCP transmission rates and the RED mechanism are more complex than for UDP. The following

result gives the expression of the average transmission rate of each user.

Proposition 2 ([4]): If all flows are TCP flows and the RED slope of each user $i \in \mathcal{I}$ is t_i, then the loss probability for a given player j is

$$p_j = t_j \frac{\left(-\mu + \sqrt{\mu^2 + 4\alpha \left(\sum_j \frac{1}{R_j\sqrt{t_j}}\right)\left(\sum_j \frac{\sqrt{t_j}}{R_j}\right)}\right)^2}{4\alpha\left(\sum_j \frac{\sqrt{t_j}}{R_j}\right)^2}.$$

Other results obained in [4] are:

- For some forms of utility functions, sufficient conditions were found under which the game converges to a strategy profile for which no user has an incentive to deviate, i.e. a Nash equilibrium.
- Parameters optimizing the network revenue for specific families of price functions d were determined numerically.

III. Game for a finite number of slopes

A. Discretized RED Game

Again, for reasons such as signalling or implementability, because not too many overhead informations can be carried on each packet, it is unlikely that a user could ask for a slope value in a continuous set. It seems indeed more realistic to choose from a finite set of predefined values (discretized RED). The strategy set available to users then becomes discrete, and the study carried out in [4] for the game played among users does not hold anymore. In particular, the determination of the Nash equilibria becomes analytically intractable, and we therefore use different tools to reach those equilibria.

We suggest in this paper to use a decentralized learning mechanism [6], [7] to reach a Nash equilibrium of the game, similarly to the approach taken in [8] in the context of wireless power control (such an approach has also been considered in [12] for the game where players select a number of TCP sessions to open). We consider that time is discretized, and suggest that users select their strategy according to a vector of probabilities that is updated at each time period of the game in order to privilege "good" strategies over "bad" ones.

Let T denote the (common) finite strategy set of each user j. User j has a probability vector $P_j(k) = (P_{j,t}(k))_{t\in T}$ (over T) for the strategy choice at k-th time step. The expected utility of user j at k-th time step given that $\forall j$, player j chooses its strategy from the probability vector $P_j(k)$ is then

$$\mathbb{E}\left[U_j(t_j, [\mathbf{t}]_{-j})\right] = \sum_{t_1,\cdots,t_I \in T} U_j(t_1, \cdots, t_I) \prod_{i=1}^{I} P_{i,t_i}(k).$$

B. Utility model for users

We assume here that each user j is sensitive to his mean throughput $\lambda_j(1-p_j)$ via a logarithmic function, i.e., user j has utility

$$U_j(t_j, [\mathbf{t}]_{-j})) = \log(1 + \lambda_j(1 - p_j(t_j, [\mathbf{t}]_{-j}))) - d(t_j) \quad (2)$$

where $d(t_j)$ is the price for slope t_j.

C. RED slopes and prices

Since there is a finite number N of different RED slopes, we number them by $1, ..., N$, such that $t^1 > t^2 > ... > t^N$. (The superscript indicates the order of the slope while the subscript is for the user index: if user j choose the kth slope then we write $t_j = t^k$). In our numerical experiments, the minimum slope t^N is determined using Theorem 1 of [4], in order to ensure that the system operates in the linear region of RED. We choose the maximum slope t^1 to be such that the discard probability tends to 1 as the virtual queue size tends to $q_{\max}$. The slopes between are equally spaced.

Prices are chosen to be of the form

$$\pi^k = d(t^k) = \underbrace{\text{price_factor}}_{=\pi} \times \sqrt{\frac{N}{N+1-k}}.$$

For example with $N = 4$, $\lambda = 20s^{-1}$, $q_{\min} = 10$ and $q_{\max} = 40$, we obtain the RED curves and prices of Figure 2.

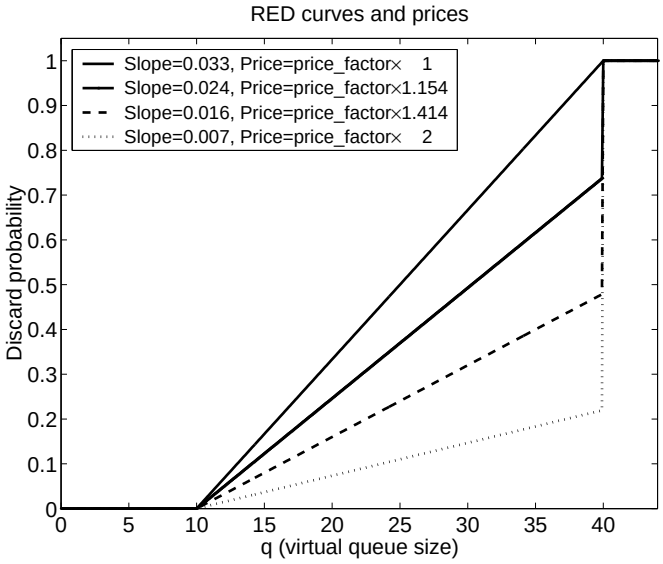


Fig. 2. RED curves and prices.

D. Bound for the Maximum Price Factor

We assume that the network users are rational in the sense that they will not try to access the network when their payoff $U_i < 0$. This imposes a maximum price factor $\pi_{\max}$ for the network operator. When this price is exceeded no network users will have any incentive to access the network. That

is, when the network operator sets $\pi > \pi_{\max}$ the network revenue will be 0, therefore an optimal price factor in terms of network revenue is smaller than $\pi_{\max}$. We give a formula for $\pi_{\max}$ in the next proposition.

Proposition 3: Let all the users have the same round-trip time R and there maximum sending rate be bounded by $\lambda_{\max}$. Then, with a discrete strategy set T, $\pi_{\max} \leq \log(1+\lambda_{\max})$.

Proof: A rational user j will be willing to access the network choosing slope $t_j = t^k$ only when $U_j(t_j, [\mathbf{t}]_{-j})) \geq 0$. Then from (2) this implies that

$$d(t_j) \leq \log(1+\lambda_j(1-p(t_j,[\mathbf{t}]_{-j}))),$$

$$i.e. \qquad \pi \leq \frac{\log(1+\lambda_j(1-p(t_j,[\mathbf{t}]_{-j})))}{\sqrt{\frac{N}{N+1-k}}}. \qquad (3)$$

Hence $\pi_{\max}$ cannot exceed the maximum value of the right-hand side of (3), which is obtained when the numerator is maximized and the denominator is minimized. Since $\sqrt{\frac{N}{N+1-k}}$ is monotonically increasing in k, its minimum is attained at $k=1$. The obvious majoration $\log(1+\lambda_j(1-p(t_j,[\mathbf{t}]_{-j}))) \leq \log(1+\lambda_{\max})$ then gives the result. ■

IV. Learning mechanism

We introduce here the user strategy decision and learning mechanism over time. The algorithm consists for each user i in updating a probability vector $P_i(s)$ at each time slot s, $P_{ik}(s)$ being the probability that user i chooses strategy (slope) t^k at instant s (the strategy t^0 corresponds to the case where the user does not establish any connection). Similarly to [8], [12], we assume that the following discrete learning algorithm is used by each player:

1) Set the initial probability vector $P_i(0)$ for each user i. In this paper we will (arbitrarily) choose uniform initial distributions over $\{0,\ldots,N\}$.
2) At each time step s, the RED slope $t_{i,s}$ is chosen by user i according to probability vector $P_i(s)$.
3) User i then monitors his throughput and determines his utility $U_{i,s}$ at time step s.
4) User i updates his probability vector according to the rule

$$P_{ik}(s+1) = \begin{cases} P_{ik}(s) - bu_{i,s}P_{ik}(s) \\ \qquad \text{if } t_{i,s} \neq t^k \\ P_{ik}(s) + bu_{i,s}\sum_{\ell \neq t_{i,s}} P_{i\ell}(s) \\ \qquad \text{otherwise.} \end{cases}$$

In words, this step consists in adjusting the probability of choosing one's strategy in the next step, considering the utility brought by the current strategy: if that utility is high ($u_{i,s}$ is large, see below) then the probability of the current strategy is increased, otherwise it is lowered.
5) If the algorithm has not converged goto step 2), otherwise stop.

In the algorithm, parameter b is the step size of the updating rule, and $u_{i,s}$ is a normalized utility

$$u_{i,s} = \frac{U_{i,s} - A_{i,s}}{B_{i,s} - A_{i,s}}$$

with $A_{i,s} = \min_{s-m\leq l\leq s} U_{i,l}$ and $B_{i,s} = \max_{s-m\leq l\leq s} U_{i,l}$ for a parameter value m that represents the memory of the user: $A_{i,s}$ (resp. $B_{i,s}$) is the minimum (resp. maximum) utility that user i obtained in the last m time slots. When $A_{i,s} = B_{i,s}$ then we set $u_{i,s} := 0$ and no update is done on the probability vector P_i.

Note that no knowledge of the number of players I is required, nor any specific knowledge of other users' strategies. Each user just needs to keep track of the best and worse utilities he recently obtained.

Again, the above game is difficult to study analytically. Note nevertheless that this is a finite game [5]. From a classical result in game theory, we know that there always exists a Nash equilibrium in mixed strategies for such a game. For general utility functions, no results of uniqueness can be given, but it is proven in [7] that when b tends to 0, the above algorithm converges to a Nash equilibrium. We therefore suggest to use that learning algorithm to discover an equilibrium of the game.

V. Simulation Results

All our simulations use the setting described in Figure 2. The memory parameter is set to 20 time slots.

A. Convergence of the algorithm

We use the above algorithm to discover some equilibria of the game in the two settings TCP users only and UDP users only.

For all the simulations we have run, it has turned out that the probability vectors for the strategy choice of each user converged to a Dirac, i.e. a strategy where the player plays a single strategy with probability 1 (pure strategy). However, having checked the resulting situation, we have observed that when the value of the parameter b in Step 4 of the learning algorithm is too large, the learned situation may not be a Nash equilibrium. This *a posteriori* verification illustrates the trade-off that relies in the choice of b, between the insurance of converging to a Nash equilibrium (when b tends to 0) and the rapidity of convergence. For our setting, a value of b around 0.01 seems to be satisfying, since it has always led to an equilibrium in our simulations.

1) Case of UDP sessions: Figure 3 shows the evolution with time of the strategies (the number of the selected slope, strategy 0 denoting the fact that the user prefers not to connect at all) chosen by the players, with the slopes and prices of Figure 2 and a price factor equal to 1. Remark that users try all the strategies at the beginning, and then rapidly tend to favour strategy 1, trying out the other ones less and less often.

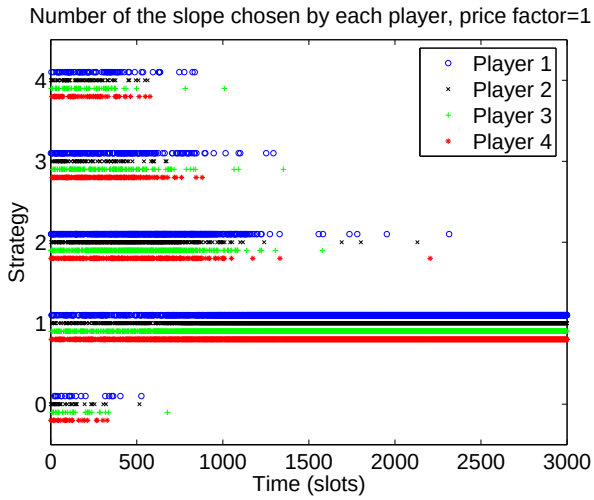


Fig. 3. Evolution of the strategies played (4 UDP users)

Figure 4 shows the evolution of the virtual queue size during the convergence phase. Since the strategies played converge, the corresponding virtual queue size also rapidly converges.

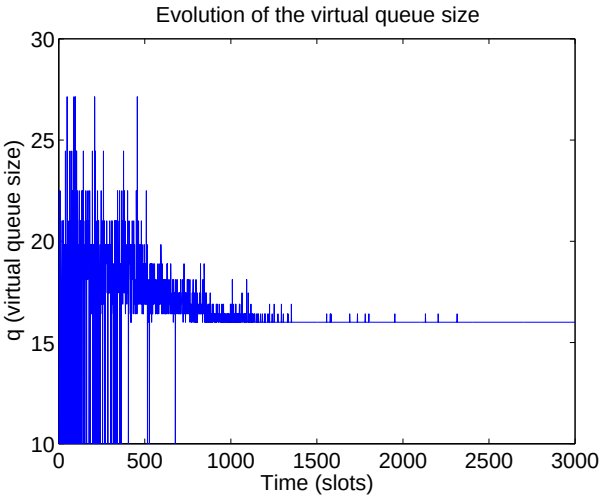


Fig. 4. Evolution of the virtual queue size (4 UDP users)

2) Case of TCP sessions: Figures 5 and 6 show the evolution of the strategies played and the virtual queue size in the case of TCP sessions (price factor=1), with similar conclusions.

B. Effect of the price factor

Due to the discontinuity of the available RED prices, the equilibria of the game are also not continuous in terms of the price factor. When the Nash equilibria are the same for different values of the price factor, then the network revenue is simply proportional to that price factor. Figures 7 and 8

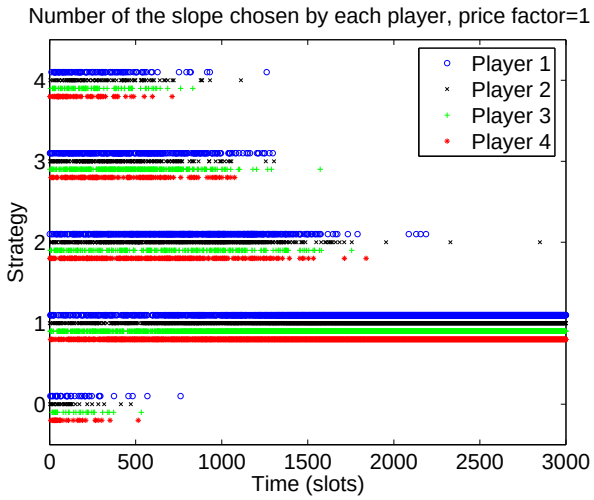


Fig. 5. Evolution of the strategies played (4 TCP users)

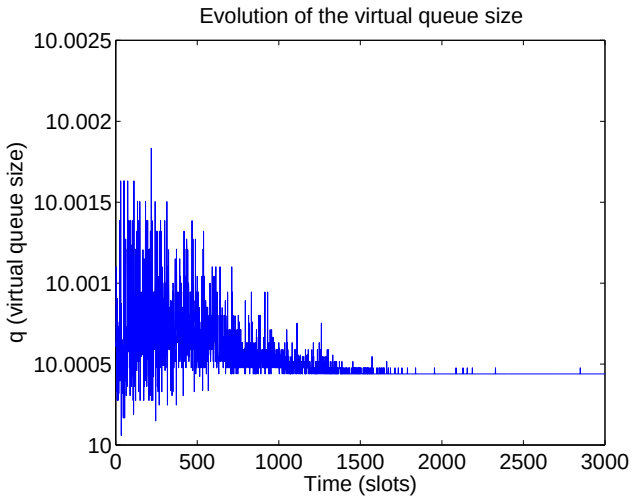


Fig. 6. Evolution of the virtual queue size (4 TCP users)

show respectively for $I = 4$ UDP users and $I = 4$ TCP users the influence of the price factor on the network revenue.

As an example, the equilibria of the game in the case of 4 UDP sessions, corresponding to Figure 7, are given in Table I (when the equilibria are not symmetric, we write the equilibria strategies - number of the slope chosen - in an ascending order). It appears that the effect of the price factor on the equilibrium is not easy to study, because the discrete nature of the game implies that the user equilibria do not change while the price factor is between two thresholds over which changes of equilibrium occur. It is therefore beneficial to the network to fix the price factor as close as possible to one of those thresholds in order to maximize his revenue. However determining analytically the optimal value of the price factor in terms of network revenue appears to be complicated since no regularity properties such as concavity

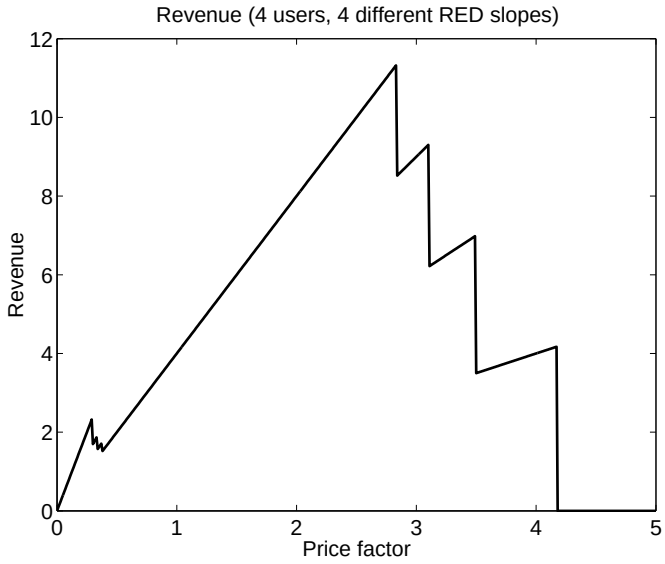


Fig. 7. Network revenue as a function of the price factor (4 UDP users)

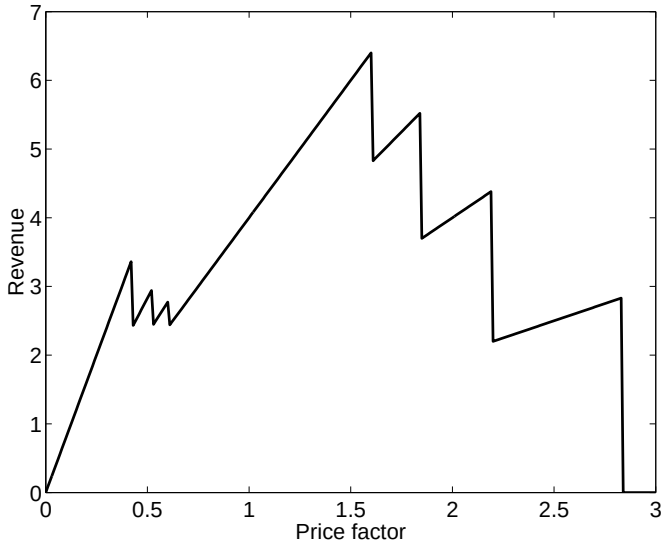


Fig. 8. Network revenue as a function of the price factor (4 TCP users)

Price factor π	Equilibrium strategies
$\pi \in [0, 0.52]$	$(4, 4, 4, 4)$
$\pi \in [0.52, 0.62]$	$(3, 3, 3, 3)$
$\pi \in [0.62, 0.71]$	$(2, 2, 2, 2)$
$\pi \in [0.71, 2.83]$	$(1, 1, 1, 1)$
$\pi \in [2.83, 3.10]$	$(0, 1, 1, 1)$
$\pi \in [3.10, 3.49]$	$(0, 0, 1, 1)$
$\pi \in [3.49, 4.17]$	$(0, 0, 0, 1)$
$\pi \in [4.17, +\infty]$	$(0, 0, 0, 0)$

TABLE I
EQUILIBRIA OF THE GAME AS A FUNCTION OF THE PRICE FACTOR π, FOR 4 UDP USERS (EACH STRATEGY PERMUTATION AMONG USERS IS ALSO AN EQUILIBRIUM).

hold, as illustrated in Figures 7 and 8. Though, a numerical evaluation can be performed as done here.

VI. CONCLUSIONS AND PERSPECTIVES

In this paper, we have presented a pricing scheme where service differentiation is provided by proposing different slopes for the RED Active Queue Management (AQM) algorithm at different prices. A finite number of choices is proposed in order to limit the signalling overhead in each packet. Analyzing the resulting game has been proved to be difficult and we have proposed a decentralized learning algorithm that converges to a Nash equilibrium among users.

We plan to work towards several directions in the future. First, we would like to study through ns-2 simulations if our fluid model is a good approximation of the packet-level and how the presented scheme can be implemented. Second, we would like to provide a more theoretical insight on the convergence of the decentralized learning algorithm. Third, we plan to investigate how such a pricing scheme could also be provided to other AQM schemes, such as gentle RED or RIO for instance.

REFERENCES

[1] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[2] P. Pieda and J. Ethridge and M. Baines and F. Shallwani, "A Network Simulator Differentiated Services Implementation," http://www.isi.edu/nsnam/ns, Tech. Rep., July 2000.

[3] "WRED," http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/wred.htm.

[4] E. Altman, D. Barman, R. El Azouzi, D. Ros, and B. Tuffin, "Pricing Differentiated Services: A Game-Theoretic Approach," *Computer Networks*, vol. 50, no. 7, pp. 982–1002, 2006.

[5] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, Cambridge, Massachusetts, 1991.

[6] K. Narendra and M. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs: Prentice Hall, 1989.

[7] P. Sastry, V. Phansalkar, and M. Thathachar, "Decentralized earning of Nash equilibria in multi-person stochastic games with incomplete information," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 24, pp. 769–777, 1994.

[8] Y. Xing and R. Chandramouli, "Stochastic learning solution for distributed discrete power control game in wireless data networks," Stevens Institute of Technology, Tech. Rep., 2004.

[9] C. Courcoubetis and R. Weber, *Pricing Communication Networks—Economics, Technology and Modelling*. Wiley, 2003.

[10] B. Tuffin, "Charging the Internet without bandwidth reservation: an overview and bibliography of mathematical approaches," *Journal of Information Science and Engineering*, vol. 19, no. 5, pp. 765–786, 2003.

[11] R. Braden, "Requirements for internet hosts - communication layers," IETF RFC 1122, Tech. Rep., Oct. 1989.

[12] B. Tuffin and P. Maillé, "How many parallel TCP sessions to open: a pricing perspective," in *5th International Workshop on Internet Charging and QoS Technology (ICQT'06)*, Saint Malo, France, June 2006, springer Verlag, Lecture Notes in Computer Science.