HAL
open science

Managing Internet routers congested links with a Kohonen-RED queue

Emmanuel Lochin, Bruno Talavera

To cite this version:

Emmanuel Lochin, Bruno Talavera. Managing Internet routers congested links with a Kohonen-RED queue. Engineering Applications of Artificial Intelligence, 2011, 24 (1), p.77-86. 10.1016/j.engappai.2010.10.012 . hal-00582608

HAL Id: hal-00582608

https://hal.science/hal-00582608v1

Submitted on 2 Apr 2011

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.

Managing Internet routers congested links with a Kohonen-RED queue

Emmanuel Lochin[1] and Bruno Talavera[2]
[1] CNRS-LAAS - ISAE, Université de Toulouse, France
[2] Université Pierre et Marie Curie, Polytech'Paris-UPMC, France
emmanuel.lochin@isae.fr, bruno.talavera@upmc.fr

Abstract

The behaviour of the TCP AIMD algorithm is known to cause queue length oscillations when congestion occurs at a router output link. Indeed, due to these queueing variations, end-to-end applications experience large delay jitter. Many studies have proposed efficient Active Queue Management (AQM) mechanisms in order to reduce queue oscillations and stabilize the queue length. These AQM attempt to improve the Random Early Detection (RED) model. Unfortunately, these enhancements do not react in a similar manner for various network conditions and are strongly sensitive to their initial setting parameters. Although this paper proposes a solution to overcome the difficulties of configuring the RED parameters by using a Kohonen neural network model, another goal of this study is to investigate whether cognitive intelligence could be placed in the core network to solve such stability problem. In our context, we use results from the neural network area to demonstrate that our proposal, named Kohonen-RED (KRED), enables a stable queue length without complex parameters setting or passive measurements to obtain a correct configuration.

1 Introduction

TCP, the predominant Internet transport protocol, has the capability to adapt its sending throughput to the changing bandwidth available following the principle described in [12]. TCP considers a loss of packets as a congestion inside the network and reacts to this congestion signal by halving its current emission window of packets. A congestion event (or loss event) corresponds to one or several losses occurring in one TCP window during one current Round Trip Time period. When a Drop Tail queue overflows,

several TCP sources can observe a bunch of losses which can lead to a decrease of the throughput involving a brutal decrease of the buffer occupancy. Then, as TCP always operates opportunistically, it restarts to increase exponentially (or linearly following the current mode is operating) in order to occupy the most capacity possible. As a result, this effect results in an accordion phenomenon where queues are constantly oscillating. This behaviour is harmful for the end-hosts which observe a constant oscillation of their current throughput that can be problematic for applications which need a relatively stable rate over the time.

More than ten years ago, the Random Early Detection (RED) was proposed to avoid congested Internet links [9]. The goal was to replace the current deployed Drop Tail queues known to provoke large queue oscillations when router's buffers overflow. The main idea of the RED algorithm is to drop packets before the queue is full. As a consequence, when a TCP source gets such preventive drops, it decreases the emitted throughput according to the AIMD (Additive Increase Multiplicative Decrease) algorithm inherent to the TCP protocol. RED drops packets with an increasing probability (max_p) when the occupancy of the queue lies between two thresholds (min_{th}, max_{th}). The goal of RED is to maintain a small buffer occupancy and avoid casual bursts of packet losses.

In [19], the authors illustrate that the instantaneous queue oscillations lead the system to a chaotic state. They show that the root of the problem comes from the estimated average queue size which is computed with inaccurate initial values. Then, during an experiment, the queue dynamics slide from a stable fixed point to an oscillatory behavior and finally to a chaotic state. As a direct consequence, the network operators cannot provide any Quality of Service (QoS) guarantee to their customers as they would need to continuously adapt their parameters for different traffic conditions.

Several other studies emphasized these issues and in particular the authors in [16] and [22] have weighted up the disadvantages for deploying such a mechanism over the Internet. In certain cases, increasing the number of dropped packets can have unexpected effects on the overall performance [22]. This has motivated the use of preventive marking instead of preventive dropping with the use of the Explicit Congestion Notification (ECN) flag [18] of IP packets. In this case, instead of dropping packets, the RED queue marks the IP packet's ECN flag to notify senders that they are crossing a congested link and that they should decrease their sending rate. Although this flag is currently implemented both in end-hosts (GNU/Linux, Mac OSX and Windows Vista and newer) and inside the core network (Cisco IOS implements a RED/ECN variant called WRED/ECN), ECN still remains disabled by

default for all these systems and the following study [17] published in 2004 precises that ECN is only used by 2,1% of computers.

Despite of all these issues, the RED algorithm is recommended by the Internet Engineering Task Force (IETF) with the expectation that the providers make their own effort to select suitable RED control parameters for their network [5]. Unfortunately, some past work have already suggested that RED is fundamentally hard to tune [14] while others claim that tuning its parameters is an inexact science [16].

All these configuration problems partly explain the reason why network operators do not enable this algorithm. Thus, it seems obvious and necessary to find a method to easily and automatically tune the RED parameters.

Several studies attempted to enhance the basic RED algorithm [6, 8, 10, 3, 14, 21, 11] . As a non-exhaustive list, Fair RED (FRED) [6] and Adaptive RED (ARED) [8] introduced the notion of adaptive AQM. These adaptive strategies recompute the max_p probability value following an AIMD algorithm. However, the parameters that weight this AIMD process still remain difficult to estimate and far from being generic as we will see in our simulation. In [14], the authors show that RED parameters can be tuned to improve stability, but only at the cost of large queues even when they are dynamically adjusted. In [21], the author introduces an interesting approach where the algorithm dynamically adjusts the moving exponential weighted computation parameter which is normally static, in order to impact on the accuracy of the rate estimation. Even if other different queueing approaches have been proposed to improve the efficiency of RED-like algorithms in various network conditions, the parameters used to set these new AQM are sometimes more complex to determine than RED ones. In particular, this is the case for the PI controller [10]. Nowadays, general parameters able to stabilize the queue do not yet exist whatever the AQM used and we could discuss whether the problem is in fact solvable.

Although the validity of RED concept is still debated, we claim that the parameters' settings are one of the main barrier to its acceptance and that this problem is a perfect candidate for the neural networking area. In this paper, we seek at evaluating whether neural network theory can help by simply acting on the standard RED parameters. As a result, this paper does not attempt to extend the prolific models previously proposed. We do not attempt to design another queueing mechanism or propose to enhance the core mechanism itself. We only focus on the optimal estimation of the probability parameter denoted max_p. Thus, this paper aims at illustrating the impact of the role of learning mechanisms on core network Internet problems with similar motivation than the one presented in [4]. The purpose

is to illustrate the perfect capability of a class of neural networks to solve this stability issue without requesting complex tuning from network engineers.

This paper is structured as follows. Section 2 presents the motivation of this work. Section 3 gives pointers related to the implementation of the core mechanism. Then, section 4 evaluates the proposal and finally section 6 gives the perspectives of this work.

2 Motivation of using a Kohonen neural network

Kohonen networks [13] are a class of neural networks known to solve the pole balancing problem [15]. Pole balancing is a control benchmark historically used in mechanical engineering. It involves a pole placed on a cart via a joint allowing movement along a single axis. The cart is able to move along a track with a fixed length as represented in Fig. 2(a). The aim of the problem is to keep this pole balanced by applying forces to the cart.

The main idea of our contribution is based on the analogy existing between this balancing problem and the RED queueing problem. In RED, we can compare the pole balancing to the evolution of the queue occupancy which oscillates between both thresholds (min_{th}, max_{th}).

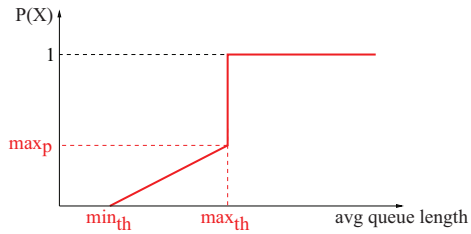
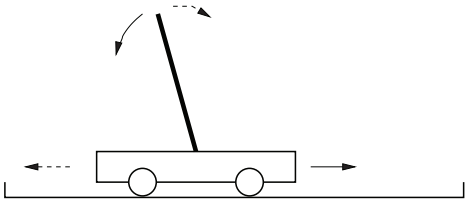


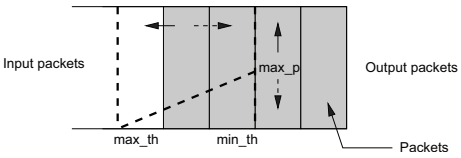
Figure 1: The Random Early Detection (RED) queue management scheme

As shown in Fig.1, the probability to drop (or to mark) packets increases when the buffer occupancy increases following the probability scheme configured (i.e. following both thresholds set and the slope of the probability curve). The physical forces resulting on the pole have a similar role to the packets arrival rate in the queue. Figure 2 illustrates this view.

Self-configuring RED schemes such as FRED or ARED update the max_p value as a function of the arrival rate in order to stabilize the queue size between both thresholds, min_{th} and max_{th}. In [6], the authors explain the queue length variation by the need of dynamically changing max_p as function of the queue occupancy. They propose to recompute this probability



(a) Pole balancing



(b) Adaptive RED

Figure 2: Analogy between the single pole balancing problem and RED AQM

	Pole	RED
input_value[1]	previous position	previous queue length
input_value[2]	new position	current queue length
output_value[1]	force to apply in Newton	max_p

Table 1: Input and output values used

following an AIMD algorithm. The update is done as function of the average queue size. If the average queue size is around max_{th}, the algorithm increases max_p to drop more packets and decreases max_p if the value is around min_{th}.

The AIMD algorithm performed by FRED is different from ARED. Indeed, FRED updates max_p each time a packet is enqueued while ARED has another parameter allowing to update this value during a time interval. This action period can smooth the effect of an aggressive setting of

the AIMD factors. Moreover, FRED does not apply consecutive decrease or consecutive increase of the max_p value. This choice can be problematic in case of rapid traffic change.

The neural network we use here is known as the Kohonen Self Organizing Map (SOM) [13]. It consists in a one or two dimensional information processing layer of functional entities called neurons. It is connected to input data seen as input vectors and provides output data also as vectors. We present in Tab. 1 the entries used to feed the neural network in both cases and the resulting output. The input vector contains the previous and the current queue length and the output vector the max_p probability. For a sake of comparison, we give in this table the vectors used with the pole balancing problem. The input data is fully mapped onto the Kohonen layer's neurons which respond to this data according to the weight assigned to the connections between input vectors and neurons and deliver an output response vector. To begin with, the neural network is presented a learning set of example input vectors and adjusts (i.e. learns) appropriate weights for its neurons by comparing the input vectors to the weight vectors for each neuron thus electing a "winning" neuron "close" to the input vector.

In addition to this, the Kohonen SOM deals with a topological learning feature, which implies neural neighborhood generalization of a correct learning experience so as to create clusters of neurons responding to similar input vectors without necessarily having explicitly learnt them. If a neuron learns that a given input vector is a vector it should respond to, its neighbours will learn they also should respond, only in a lesser way, depending on their topological distance to the first "winning" neuron. This way, the Kohonen SOM is well adapted to stability preservation tasks as the one we present here. Once the learning procedure is over, i.e. when the neural network produces an acceptable amount of erroneous responses during learning, the weights of the neural connections to the data input are frozen. That means that the training process needs to be done only once without specific scenario and should work for every kind of situation.

Given the Kohonen SOM algorithm, the neural network can generalize its learnt experiences to other input vectors it has never seen before and produce adapted responses. In this way, the conservation of a direction, an equilibrium or the correct parameter to adjust a RED mechanism is made possible although there is no way of predicting the way the neural network learns to solve this particular problem. In our case, the learnt sequences of input vectors are not the ones used in our tests, in order to prove that the learning method provides a general purpose neural network for the resolution of the problem we deal with here. Once it has learnt, it

can be used indefinitely for the task it has been trained for.

Previous related work [7] presents the use of a multi-layer perceptron to adapt the α and β coefficients of a PI controller. In [7], the authors don't improve the queue length stability but smooth the PI dynamic and in average, results obtained are globally similar. We think that such a neural network is well adapted to pattern and shape recognition problems, whilst a SOM such as the Kohonen SOM could be better suited to the task of stability preservation which we deal with here. Indeed, this Kohonen SOM algorithm preserves topological relationships between neighbouring vectors.

Each time a packet is enqueued, the Kohonen network computes a new max_p following the previous and the current average queue size. No other parameters are needed to perform this operation.

3 Implementation

One important point of dealing with Kohonen networks is the small memory footprint required by the implementation. In our case, we have implemented our proposal in ns-2 simulator [1]. The most complex structure is simply a square matrix 25×25 which represents the Kohonen network. The code used is a modification of the well-known Karsten Kutza's implementation[1]. For the sake of comparison, all the scripts and ns-2 implementation used in this study are available for download at the author's webpage[2].

4 Evaluation and analysis

This section presents the experiments driven to evaluate the KRED and comments the results obtained. For the sake of comparison purpose, we base our hypothesis and initial RED parameters on this following recent and exhaustive study [11]. This allows the reader to better compare our results with existing work and simulations realized in this area as well as avoiding any fancy or unjustified use of initial RED parameters.

4.1 Testbed and assumptions

We drive experiments over a standard dumbbell topology represented in Fig. 3. We compare our proposal to RED, Fair RED (FRED), ARED, Random Exponential Marking (REM) [3] and Proportional Integral (PI) AQM. The

[1] http://www.neural-networks-at-your-fingertips.com/
[2] http://manu.lochin.net/kred

parameters used for each queue are given in table 2. For each queue, every threshold is set in order to obtain a stabilization around half the queue size.

The parameters given Tab. 2 come from standard recommendation from the networking community and are recalled in our reference paper [11]. All these parameters have been deeply investigated by the past and are consistent with the following study. Furthermore, the testbed and the networking conditions have not been chosen randomly, they strictly follow [11] in order to allow a fair comparison with the results presented.

The TCP flows are NewReno with a large window size set to 10000 packets. The RED queue is configured to drop and not to mark packets. In order to evaluate our proposal, we drive two distinct experiments. In the first scenario, the number of TCP flows in the network is increasing from 50 to 250 flows following the pattern Fig. 4(a). The RTT for each flow is identical. This scenario allows us to verify the impact of the traffic load on our proposal compared to other AQMs. In the second experiment, the traffic changes every 50 seconds following the scenario presented in Fig. 4(b). The rationale for using this traffic pattern is to evaluate our proposal under wide traffic variations.

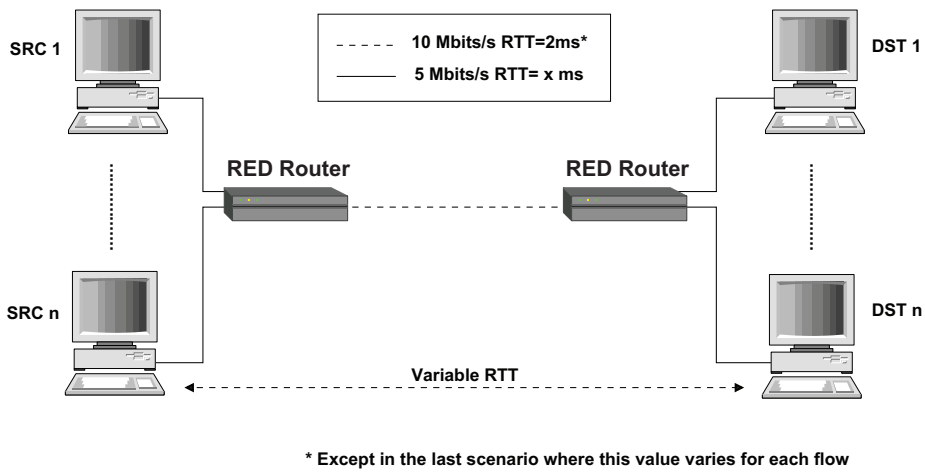
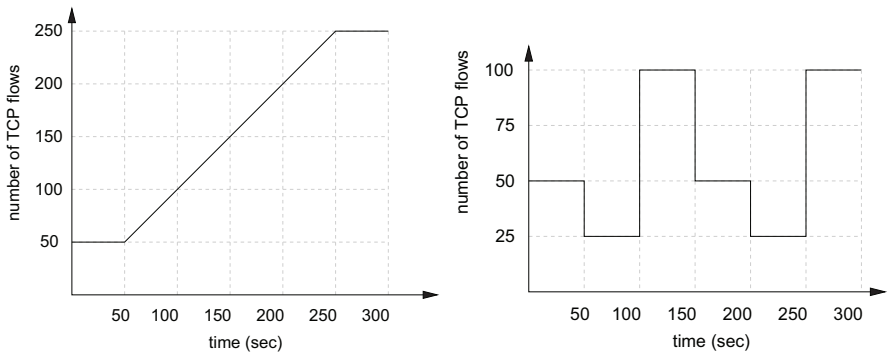


Figure 3: The simulation topology

Common Parameters (C.P.)	$q_{size} = 200pkts$ $q_{weight} = 0.00008$
RED	C.P., $max_p = 0.1$, $gentle = true$, $min_{th} = 0.25 * q_{size}$ $max_{th} = 0.75 * q_{size}$
FRED	C.P., $max_p = 0.1$ $\alpha = 3.0$, $\beta = 2.0$
ARED	C.P., $\alpha = 0.01$, $\beta = 0.09$, $gentle = true$, $interval = 0.3$, $max_p = 0.1$, $min_{th} = 0.25 * q_{size}$ $max_{th} = 0.75 * q_{size}$
REM	$\gamma = 0.001$, $\phi = 1.001$, $\alpha = 0.1$, $interval = 2ms$, $q_{ref} = 0.5 * q_{size}$
PI	$a = 1.822.10^{-5}$, $b = 1.816.10^{-5}$, $q_{ref} = 0.5 * q_{size}$, $w = 170Hz$
KRED	C.P., $min_{th} = 100pkts$ $max_{th} = 150pkts$

Table 2: RED parameters used



(a) First scenario (b) Second scenario

Figure 4: The simulation scenarios

4.2 The training process

The KRED queue has been trained with an arbitrarily chosen number of twenty long-lived TCP/Newreno flows emitted during 600 seconds without any traffic variation on a single link topology. The delay of the core link was set to 50ms which corresponds to an RTT of 100ms. We choose this RTT following this measurement study [2] which shows that the global RTT observed over the Internet is equally spread around 100ms over an interval ranging from 1ms to 1sec. The neural network map learnt to stabilize the KRED queue with the KRED parameters given Tab. 2 after 179 seconds. No further training has been done.

The learning process is automatically stopped when the relative error tends towards zero (i.e. when the queue value is correctly stabilized between the two requested bounds). The scenario used to train the neural network is the most simple one as no traffic variation is introduced. We attempted to train KRED following a scenario with several traffic variations (as 4(b)). The training process in that case was much longer (around 500 seconds). When we experimented the resulting Kohonen map, we did not obtain better results for scenario #2 with this training (in fact, results where quite similar) however, the Kohonen map did not perform well with scenario #1 or any other scenarios involving other traffic patterns. We concluded that our resulting Kohonen map was overspecialized and decided to come back to a simple training scenario. The best and the more generic map was obtained by training the neural network with a relatively small number of TCP sources without introducing any traffic variations. As a result, the resulting Kohonen map is used thereafter in all experiments. We also have driven other measurements, not presented in this paper, and clearly obtained the same results with other traffic variation.

4.3 First scenario

Results are given in Fig. 5. Each graph presents the instantaneous and the average queue size. The results presented for the KRED queue are obtained after the training process. As shown in these figures, ARED 5(c) and KRED 5(f) queues obtain a stable queue length between both thresholds compared to the other queues. To better illustrate the performance of our algorithm, we compare the queue delay and the average drop rate of all these proposals in table 3. This table clearly shows that both ARED and KRED obtain similar performances in terms of drop rate ratio and queue delay. We remark that we haven't reported in this table the link usage obtained by each queue

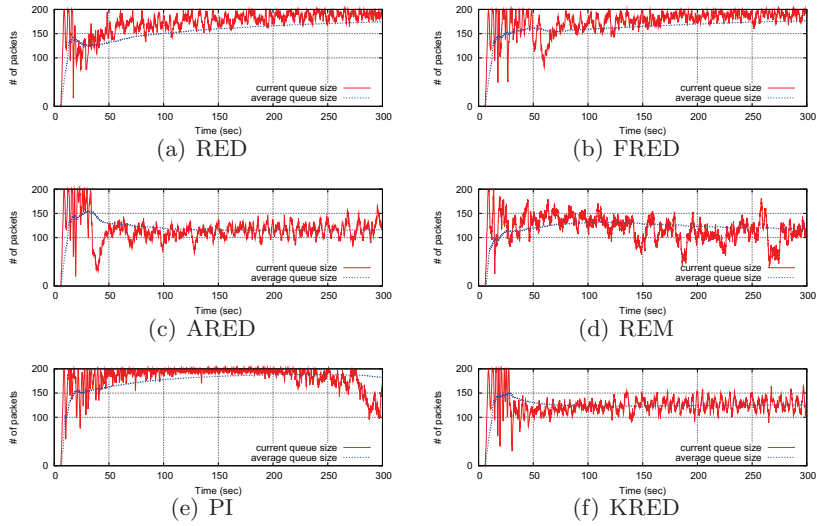


Figure 5: Performance comparison of various AQM with KRED, (1st scenario)

as the latter is equal to 4.96Mbit/s with a nil standard deviation for all the experiments.

As in [11], it seems that ARED is the best competitor, we then compare ARED to KRED with various end-to-end RTT. The results are presented in Fig. 6. This figure globally shows that KRED and RED stabilizes the queue around the target value of 100 packets whatever the RTT of the network. However, we can notice a slight advantage for KRED which is more stable when the RTT increases.

With this first experiment, we can conclude that the overall performances of ARED and KRED are similar in terms of stability of the queue when the traffic load increases. As a result, both initial ARED parameters (given in table 2) which tune the aggressiveness of the additive increase and multiplicative decrease drop probability value and KRED algorithm reach their goal.

Thus the second scenario presented in the next section extends these measurements by changing the traffic pattern during the simulation. The original parameters remain unchanged in order to assess the adaptability of these AQM to the rapid traffic change conditions.

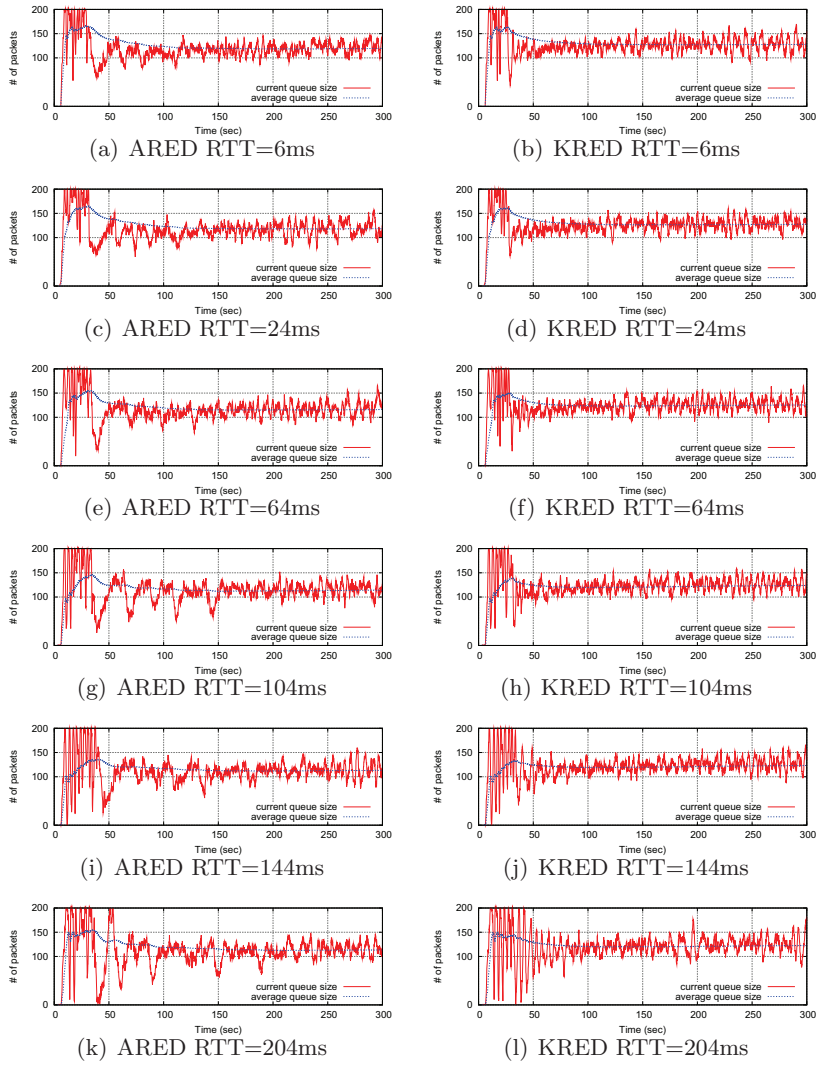
AQM	Mean / Std. Dev. Queue Delay (ms)	TCP drop rate
RED	22.52 / 7.55	8.42%
FRED	19.64 / 8.72	10.21%
ARED	18.30 / 7.46	10.72%
REM	18.54 / 8.52	10.29%
PI	22.05 / 10.53	9.35%
KRED	19.11 / 4.42	10.50%

Table 3: Statistics from Fig. 5

4.4 Second scenario

As shown in Fig. 7, the KRED queue obtains a stable queue length between both thresholds compared to the AIMD process method of the FRED 7(b) and ARED 7(c) algorithms. Furthermore, KRED reacts rapidly to the traffic change compared to ARED. Due to this fast traffic changing, the max_p value is constantly recomputed and the previous computed value strongly impacts on the current result. In the case of an AIMD process which estimates the max_p value, if the weights are small (i.e. α and β parameters of table 2), the pace of convergence to the optimal value is slow and if the weights are high, the resulting probability can strongly oscillate when the traffic is changing. As a consequence, the initial configuration parameters used with success in the first scenario by ARED are not adapted to the second one. Thus, our proposal allows to overcome this difficult problem of initial setting which is perfectly managed by the Kohonen neural network. Finally, table 4 gives the statistics of this scenario and shows that KRED obtains stable queue length without average queuing delay compromise.

In Fig. 8, we also change the nominal RTT value between all hosts. The immediate observation is that KRED reacts much faster to the traffic variation. However, when the queue is slightly loaded (between $t = [50; 100]$ and $t = [200; 250]$) and the RTT is high, KRED gets some difficulties to stabilize the queue. Indeed, we do not take into consideration as in [11] the evolution of the RTT as a parameter to impact on the dropping probability. As this system is a control feedback loop system, the reaction of the sources is delayed an RTT later. In case of high load, the high number of multiplexed TCP sources allows to increase the number of clients that conjointly react to the congestion signal (i.e. the packets prevently dropped by the queue). Obviously, if we take into account the RTT of each flow and that we consider



(a) ARED RTT=6ms (b) KRED RTT=6ms

(c) ARED RTT=24ms (d) KRED RTT=24ms

(e) ARED RTT=64ms (f) KRED RTT=64ms

(g) ARED RTT=104ms (h) KRED RTT=104ms

(i) ARED RTT=144ms (j) KRED RTT=144ms

(k) ARED RTT=204ms (l) KRED RTT=204ms

Figure 6: Performance comparison of ARED and KRED with various RTT, (1st scenario)

an average value of the RTT evolution, we could add to KRED another useful entry parameter which indicates to the system the minimum reaction delay. However, realizing such passive measurements of the RTT is quite complex and allowing the queue to assess the RTT of each flow is not realistic. Indeed,

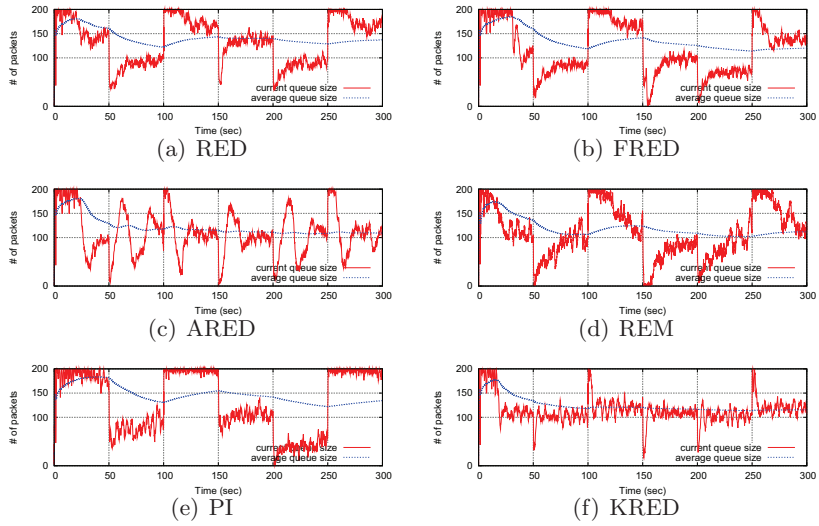


Figure 7: Performance comparison of various AQM with KRED (2nd scenario) with a nominal delay queue of 30ms

this supposes maintaining a per-flow state inside the router and greatly complexifies the core router internal computation. This violates the well-known Internet paradigm which states that *the intelligence and processing power of a network must reside at the outer edges while the inner network itself remains as simple as possible* [20]. In our context, the goal is to assess if our proposal is generic enough to be used in a broad context compared to other competitive proposals that involve complex setting parameters. As illustrated in Fig. 8, KRED solves a broader range of network conditions compared to other proposals.

4.5 Second scenario with variable RTT

In this last experiment, we still use the second experiment scenario where the traffic changes every 50 seconds but in addition to this basic scenario, each flow has a random RTT ranging from x to $x + 38$ms. This corresponds to a variable RTT of the access link which is ranging from 1 to 20ms of delay.

The purpose is to simulate a more realistic traffic behaviour to assess the performance of KRED in a general context and in particular, if we consider

AQM	Mean / Std. Dev. Queue Delay (ms)	TCP drop rate
RED	29.64 / 4.21	27.25%
FRED	29.86 / 4.08	27.40%
ARED	19.58 / 3.95	30.53%
REM	18.15 / 4.73	31.05%
PI	25.39 / 7.55	28.65%
KRED	20.96 / 3.39	30.15%

Table 4: Statistics from Fig. 7

AQM	Mean / Std. Dev. Queue Delay (ms)	TCP drop rate
RED	21.01 / 7.72	7.05%
FRED	21.47 / 8.31	7.50%
ARED	17.92 / 8.71	9.27%
REM	18.33 / 8.01	8.53%
PI	21.10 / 11.02	7.79%
KRED	18.79 / 4.53	8.74%

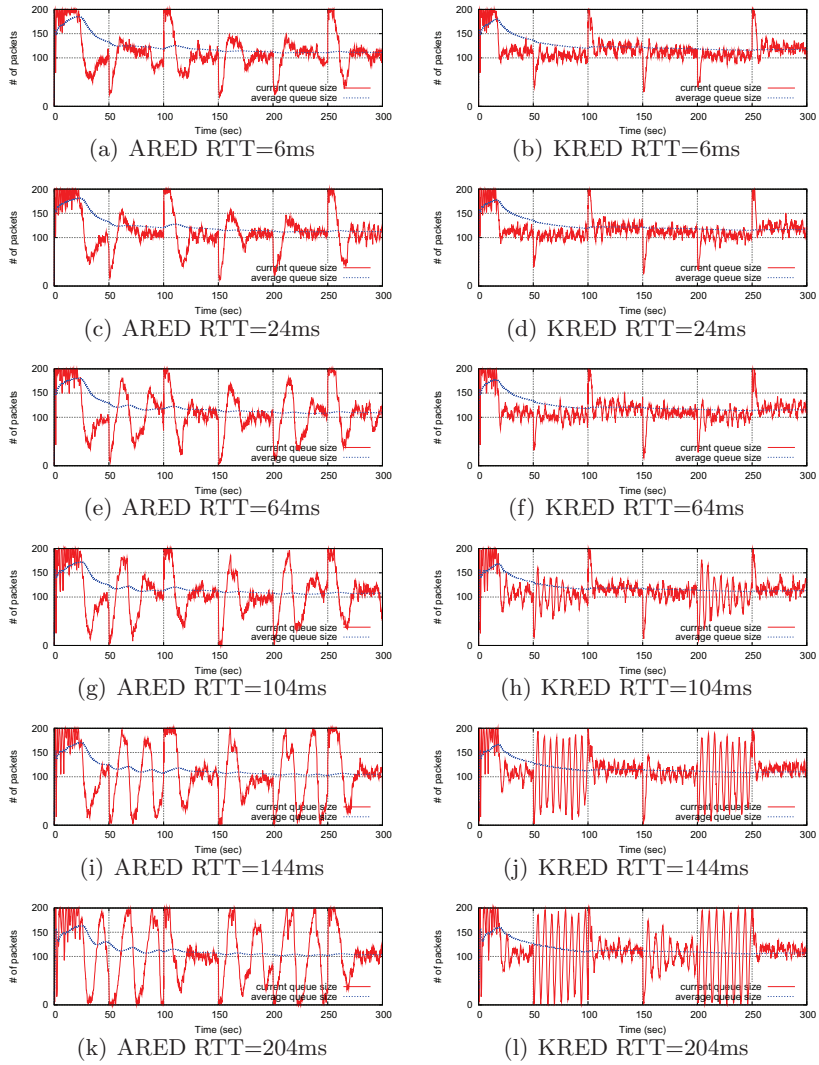
Table 5: Statistics Fig. 9

the active queue placed near the edge or inside the core network. Then, using various RTT ranges, we illustrate several localization contexts. We do not claim that the scenario proposed perfectly simulates a real topology, we just want to illustrate the versatility of our proposal

Fig. 9 and table 5 report the results obtained for each queue where each flow gets an RTT randomly chosen between $[104, 142]$ ms. Once again, KRED and ARED seem the most efficient queues. If we investigate the case of smaller and higher RTT ranges in Fig. 10, KRED remains the most versatile queue.

5 Statistical analysis

To improve the confidence and better interpret the results obtained, we have computed the mean, standard deviation and used a metric commonly-used in the control theory area to estimate the stability of our mechanism. The so called "out of range" metric gives the percentage of instantaneous values



(a) ARED RTT=6ms (b) KRED RTT=6ms

(c) ARED RTT=24ms (d) KRED RTT=24ms

(e) ARED RTT=64ms (f) KRED RTT=64ms

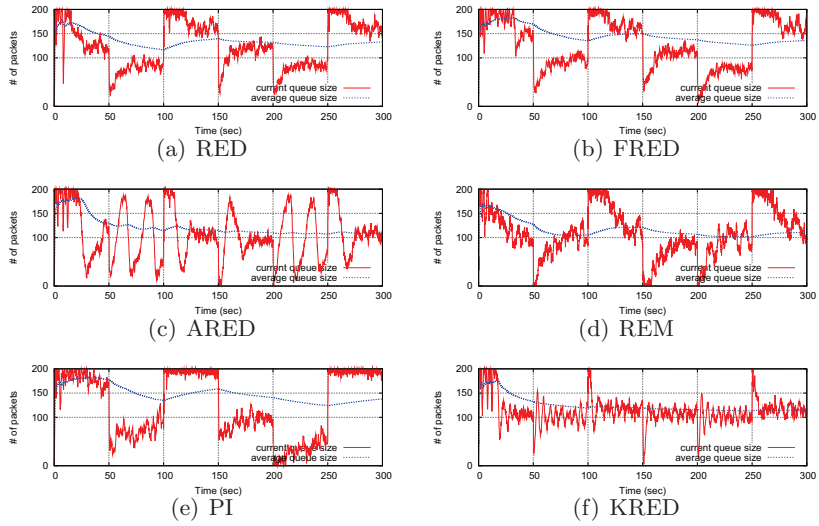
(g) ARED RTT=104ms (h) KRED RTT=104ms

(i) ARED RTT=144ms (j) KRED RTT=144ms

(k) ARED RTT=204ms (l) KRED RTT=204ms

Figure 8: Performance comparison of ARED and KRED with various nominal RTT, (2nd scenario)

inside and outside an envelop defined between $\pm 20\%$ of the computed mean. This metric allows to assess a relative stability of both queueing disciplines. As each computed mean is different, we have also computed the coefficient of variation (also known as relative standard deviation which corresponds



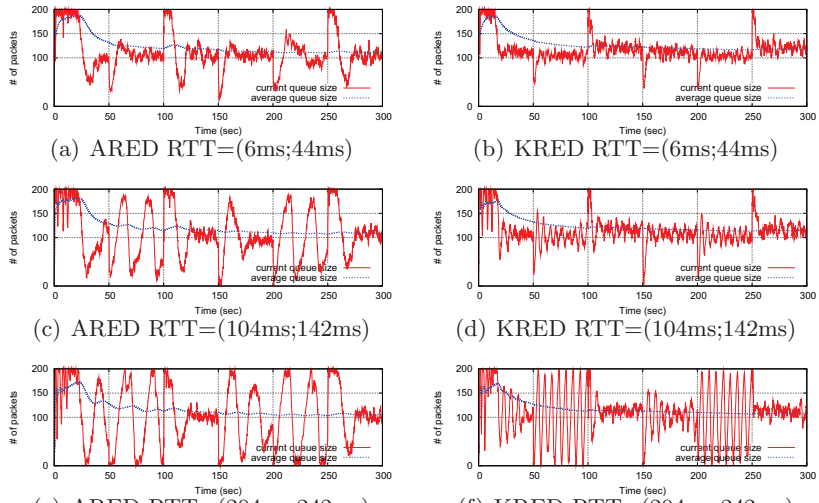
(a) RED (b) FRED (c) ARED (d) REM (e) PI (f) KRED

Figure 9: Performance comparison of various AQM with KRED with a variable RTT for each flow (2nd scenario) ranging from 104ms to 142ms

to the ratio between the standard deviation and the mean) to ease the comparison. Tables 6, 7, 8 provide the resulting statistics respectively for Fig. 6, 8, 10. All the statistics have been computed starting at t=50 seconds to suppress the non-steady phase. All these results show that the dispersion is always smaller for KRED. Clearly, the best scores are always obtained by KRED. Although we can observe a slight advantage for KRED in Tab. 6, the results obtained by ARED are in the same order of magnitude. This is explained by the scenario #1 which performs smooth changes (we recall that in this scenario the load is gradually increased) compared to scenario #2 where more abrupt changes occur, thus preventing the correct adaptability of ARED algorithm.

6 Conclusion

This paper introduces Kohonen-RED: an adaptive RED mechanism easily implementable. The idea deals with the use of a Kohonen neural network to compute the optimal probability parameter in order to achieve a stable queue length. KRED reduces the number of parameter settings and in particular the non obvious ones. The Kohonen network does not need to be



(a) ARED RTT=(6ms;44ms) (b) KRED RTT=(6ms;44ms)

(c) ARED RTT=(104ms;142ms) (d) KRED RTT=(104ms;142ms)

(e) ARED RTT=(204ms;242ms) (f) KRED RTT=(204ms;242ms)

Figure 10: Performance comparison of ARED and KRED with a variable RTT for each flow (2nd scenario)

Experiment	Mean	StDev	CV	Out of range (%)
KRED RTT=6ms	128.2291	13.7077	0.10	6.9029
ARED RTT=6ms	119.7281	14.6109	0.12	9.7330
KRED RTT=24ms	128.5940	13.3780	0.10	5.0511
ARED RTT=24ms	121.0235	14.1784	0.12	8.0824
KRED RTT=64ms	127.0030	13.0313	0.10	4.1380
ARED RTT=64ms	117.9092	15.3494	0.13	12.0943
KRED RTT=104ms	125.9923	14.3782	0.11	7.7991
ARED RTT=104ms	115.7074	16.7704	0.14	14.6647
KRED RTT=144ms	125.1373	14.6045	0.11	8.7754
ARED RTT=144ms	114.1829	14.8270	0.13	11.1980
KRED RTT=204ms	124.1143	17.8153	0.14	16.0921
ARED RTT=204ms	113.7220	18.7730	0.16	16.5810

Table 6: Resulting statistics for scenario 1 corresponding to Fig. 6

retrained and therefore can be put in hardware in the context of a router implementation. The mechanism's efficiency has been illustrated through ns-2 simulation where other schemes fail. In this work, we use a Kohonen based

Experiment	Mean	StDev	CV	Out of range (%)
KRED RTT=6ms	113.8994	21.1866	0.18	11.3929
ARED RTT=6ms	106.9419	33.3983	0.31	35.4935
KRED RTT=24ms	113.4426	21.1925	0.18	10.0620
ARED RTT=24ms	108.5999	34.4630	0.31	33.5782
KRED RTT=64ms	112.0166	22.1133	0.19	13.6567
ARED RTT=64ms	105.4034	41.0358	0.38	50.7675
KRED RTT=104ms	109.8805	25.5262	0.23	24.6657
ARED RTT=104ms	104.3858	43.4935	0.41	51.8414
KRED RTT=144ms	106.5862	35.8957	0.33	37.3802
ARED RTT=144ms	101.4978	54.7824	0.53	63.1500
KRED RTT=204ms	103.2339	42.7257	0.41	47.2238
ARED RTT=204ms	99.0713	56.6253	0.57	66.9472

Table 7: Resulting statistics for scenario 2 corresponding to Fig. 8

Experiment	Mean	StDev	CV	Out of range (%)
KRED RTT=(6ms;44ms)	114.4116	19.2592	0.16	9.4419
ARED RTT=(6ms;44ms)	106.4614	33.9947	0.32	30.7016
KRED RTT=(104ms;142ms)	110.7031	22.2922	0.20	17.5226
ARED RTT=(104ms;142ms)	104.3362	48.6995	0.46	56.0970
KRED RTT=(204ms;242ms)	104.0744	39.5693	0.38	41.1171
ARED RTT=(204ms;242ms)	100.3485	57.4776	0.57	65.3968

Table 8: Resulting statistics for scenario with variable RTT corresponding to Fig. 10

neural network specifically designed to solve the pole balancing problem. One of the main contribution of this study is to show the feasibility of using neural networks to solve a networking stability problem and the versatility of such a mechanism. We believe that specific training can be realized as a function of the queue characteristics (i.e. output link throughput, queue size, ...) in order to provide several maps to fit several contexts.

Now, considering other engineering domains, we know this neural network can solve several stability problems that belong to mechanical engineering. Today and to the best of our knowledge, this application to networking problems is original. As a future work, we are still investigating this class of neural network to assess whether it could help to stabilize a TCP

sender throughput or could be used in other networking issues. Finally and following these promising preliminary results, we are currently evaluating other neural network algorithms and considering the development of this algorithm inside the GNU/Linux kernel.

Acknowledgments

The authors would like to thank Sebastien Ardon and Guillaume Jourjon and Max Ott for the discussion about this mechanism and the support of the National ICT Australia (NICTA).

References

[1] ns-2 network simulator.

[2] Jay Aikat, Jasleen Kaur, Donelson F. Smith, and Kevin Jeffay. Variability in tcp round-trip times. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, Miami Beach, FL, USA, 2003.

[3] Sanjeewa Athuraliya, Victor H. Li, Steven H. Low, and Qinghe Yin. Rem: Active queue management. *IEEE Network*, 15(3):48–53, May 2001.

[4] Robert Beverly and Karen Sollins. The role of learning in network architecture, 2007. Research Abstract of the Computer Science and Artificial Intelligence Laboratory (CSAIL) - http://publications.csail.mit.edu/abstracts/abstracts07/beverly2/beverly2.html.

[5] Bob Braden, David D. Clark, Jon Crowcroft, Bruce Davie, Steve Deering, Deborah Estrin, Sally Floyd, Van Jacobson, Greg Minshall, Craig Partridge, Larry Peterson, K. K. Ramakrishnan, Scott Shenker, John Wroclawski, and Lixia Zhang. Recommendations on queue management and congestion avoidance in the Internet. Request For Comments 2309, IETF, April 1998.

[6] Wu chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G. Shin. A self-configuring RED gateway. In *Proceedings of INFOCOM 99*, volume 3, pages 1320–1328, 1999.

[7] H.C. Cho, M.S. Fadali, and Hyunjeong Lee. Neural network control for tcp network congestion. In *Proc. of the 2005 American Control Conference*, pages 3480–3485, June 2005.

[8] S. Floyd, R. Gummadi, and S. Shenker. Adaptive red: An algorithm for increasing the robustness of red, technical report, international computer science institute, August 2001.

[9] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[10] C. V. Hollot, Vishal Misra, Donald F. Towsley, and Weibo Gong. On designing improved controllers for AQM routers supporting TCP flows. In *INFOCOM*, pages 1726–1734, 2001.

[11] Jong hwan Kim and Ikjun Yeom. Reducing queue oscillation at a congested link. *IEEE Transactions on Parallel and Distributed Systems*, 19(3):394–407, 2008.

[12] Van Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM*, pages 314–329, Stanford, CA, August 1988.

[13] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Information Sciences*. Third extended edition edition, 2001.

[14] Steven H. Low, Fernando Paganini, Jiantao Wang, and John C. Doyle. Linear stability of tcp/red and a scalable control. *Computer Networks*, 43(5):633–647, 2003.

[15] A. Makarovic. Machine intelligence 12: towards an automated logic of human thought. In *Clarendon Press, New York, NY, USA*, pages 241–258, 1991.

[16] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In *Proc. of 7th. International Workshop on Quality of Service (IWQoS'99), London*, pages 260–262, June 1999.

[17] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the internet. *Computer Communication Review*, 35(2), April 2005.

[18] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP, September 2001.

[19] Priya Ranjan, Eyad H. Abed, and Richard J. La. Nonlinear instabilities in TCP-RED. *IEEE/ACM Transactions on Networking*, 12(6):1079–1092, 2004.

[20] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2.

[21] Shan Suthaharan. Reduction of queue oscillation in the next generation internet routers. *Computer Communications*, 30(18):3881–3891, 2007.

[22] T. Ziegler, S. Fdida, and C. Brandauer. Stability criteria of RED with TCP traffic. In *IFIP ATM&IP Working Conference*, Budapest, June 2001.