Influence of Active Queue Management Parameters on Aggregate Traffic Performance

Martin May, Christophe Diot, Bryan Lyles, Jean Bolot

INRIA
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Influence of Active Queue Management Parameters on Aggregate Traffic Performance

Martin May — Christophe Diot — Bryan Lyles — Jean Bolot

N° 3995

Août 2000

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--3995--FR+ENG

INRIA
SOPHIA ANTIPOLIS

Influence of Active Queue Management Parameters on Aggregate Traffic Performance

Martin May*, Christophe Diot† , Bryan Lyles‡ , Jean Bolot§

Thème 2 —Génie logiciel
et calcul symbolique
Projets Rodeo and Sprint IP Research Group

Rapport de recherche n° 3995 —Août 2000 —21 pages

Abstract: Active queue management (AQM) is a family of packet dropping mechanisms for FIFO queues that have been proposed to support end-to-end congestion control mechanisms in the Internet. In this paper, we examine the four possible combinations of queue size averaging vs instantaneous queue size and a sharp dropping function vs a smooth dropping function. The combinations include both the well known "Drop from Tail" (TD) and "RED" mechanisms as well as the more recently proposed "Gentle RED" and a previously unrecognized mechanism we call "Gentle RED with instantaneous queue size", or GRED-I.

The results show the end-to-end performance of the four AQM mechanisms. Thus, we do not study individual TCP flow performance but instead the flow aggregation behavior through backbone nodes, and we chose experimental parameters and observed metrics accordingly.

We show that if TD and RED have similar behaviors in terms of aggregate TCP throughput, RED discards more non responsive traffic than TD. On the other hand, we show that RED creates more consecutive drops than TD. We identify queue size averaging and the slope of the dropping function as the two major parameters influencing consecutive drops.

Key-words: Queue Management, RED, Performance Evaluation, Experimental Studies

* INRIA Sophia Antipolis
† Sprintlabs Burlingame
‡ Sprintlabs Burlingame
§ ENSIM

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 93 65 77 77 - International : +33 4 93 65 77 77 —Fax : 04 93 65 77 65 - International : +33 4 93 65 77 65
à partir du 01/01/1998
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 —Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

L’impact de la gestion de fils d’attends active sur la performance du trafic aggrege

Résumé : La gestion active de files d’attente consiste en un ensemble de mécanismes de rejet de paquets proposés pour améliorer le contrôle de congestion de bout-en-bout dans l’Internet.

Dans ce papier, nous étudions et comparons l’utilisation de la taille moyenne ou instantanée de la file d’attente combinée avec une fonction de rejet en escalier ou une fonction de rejet lisse. Parmi ces quatres combinaisons possibles, on retrouve les mécanismes actuellement utilisés dans l’Internet : Tail Drop (TD), RED (Random Early Detection), ainsi qu’une variante proposée recemment “gentle RED” (GRED). Nous examinons aussi une nouvelle forme de GRED utilisant la taille instantanée de la file d’attente : GRED-I.

Les résultats montrent la performance de bout-en-bout des quatre mécanismes. Nous ne nous interressons pas ici à la performance des flux individuels mais au comportement d’un aggrégat de flux.

Nous montrons que si RED et TD se comportent de fa con similaire en terme de débit aggregé TCP, RED créé plus de pertes consecutives que TD. L’utilisation de la taille moyenne de la file d’attente et la forme de la fonction de rejet sont les deux paramètres majeurs qui influencent les pertes consecutives.

Mots-clés : contrôle de congestion, contrôle des files l’attente, RED

I. INTRODUCTION

Active queue management (AQM) is a family of packet dropping mechanisms for FIFO queues that have been proposed to support end-to-end congestion control mechanisms in the Internet. AQM is based on a very simple principle: implicit feedback of the aggregate to network hosts by dropping packets at router queues. The network hosts then react to the packet loss by reducing the amount of data sent [1]. The AQM mechanisms that we are considering deal with aggregations of flows, and do not provide feedback to TCP or UDP users based on the congestion status of the individual flows. Per-flow queueing is another family of control mechanisms which does provide feedback based on individual flow status [2] but which is viewed as being significantly more complex and is thus generally not supported in operational networks. Until RED was proposed by the end-to-end IRTF group [3], Tail Drop (or Drop from Tail) was the only mechanism used in network nodes to control congestion. Tail Drop remains today the most popular mechanism in IP routers, mostly because of its robustness and because of its simple implementation.

The RED scheme was initially described and analyzed in [4]. RED enhances Tail Drop by introducing two new elements in the queue management scheme:

- A *dropping function* that instead of waiting until the queue is full to drop packets (as in Tail Drop) anticipates, and hopefully avoids, congestion by beginning to drop a percentage of the packets earlier than strictly needed. The parameters of the dropping function were first defined in [4] and later modified in [5] and [6]. The first modification mostly adjusted the RED parameter values. The second modification [6] was a major change that introduced a new parameter in the dropping function. Figure 1 illustrates the evolution of the RED dropping function. In this paper we call the AQM mechanism described in [4] and [5] "RED", and we call the mechanism described in [6] "Gentle RED".
- The dropping probability is adjusted based on the *averaged queue size* of the router, rather than the instantaneous queue size as in Tail Drop. The average queue size is estimated using an exponential weighted moving average :

$$\hat{k} \leftarrow (1-w)\hat{k} + w\ k,$$

Where w is a fixed (small) parameter and k is the instantaneous queue size. The value of w determines the amount of "memory" that is taken into account to calculate the average queue size. The average queue size is intended to smooth the router's reaction to traffic increases and decreases. Note, that the calculation of the average queue size increases the complexity of the buffer management algorithm.

The left function in figure 1 represents the function used in [4]. After further simulations, it was necessary to increase the maximal drop probability from $max_p = 0.02$ to $max_p = 0.1$ because the 2% drop probability was not enough to force multiple TCP sources to sufficiently reduce their window sizes [5]. However, the simulations of the RED algorithm did not reflect the traffic that is mostly found today on an IP backbone network. In a situation where thousands of flows compete for buffer space, a few packet discards could not regulate the traffic sources since at least one discard is needed per flow in order to provide feedback to TCP. Hence, in [6], the authors define a drop function that increases slowly from 0 to 100%.

Despite the adulation around RED, there are still very few published detailed performance studies of it. There is also little reported operational experience of RED in large scale networks. One

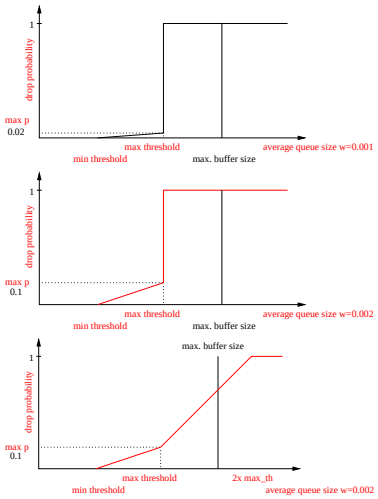

Fig. 1
THE EVOLUTION OF THE RED DROPPING FUNCTIONS

of the few published measurement study is limited in scope because it only considers the router performance (as opposed to the end-to-end performance) and it does not clearly describe the measurement settings and the exact information being measured [7]. In other publications, the authors use simulations to evaluate RED and examine the impact of the parameter choice on the end-to-end performance [8], [9], [10], [11], [12], [13]. But, realistic simulation settings are hard to come up with *, and the traffic generated by most simulators is generally different from real network traffic. Simulations typically use infinite greedy TCP sources, connections with constant round trip delays, or a small number of connections. A couple of papers have been published that propose extensions or changes to the RED algorithms to make it more robust or adaptive like FRED [14], or SRED [15], however these papers do not question or evaluate the performance and suitability of the RED algorithm. Consequently, it is not yet clear how to choose AQM parameters. The only parameter values can be found in [5] and [6]. These values also rely on simple experiments as well as on the authors' intuition.

In this paper, we characterize AQM mechanisms using the two parameters introduced above: the shape of the dropping function and the average queue size. We propose an analysis of the design space of AQM based on these parameters. Table I summarizes the AQM design space. We analyze first Tail Drop and RED. Then we discuss the two other mechanisms (lower line of the table), namely GRED and GRED-I (the latter one will be explained below).

* They require choosing many parameters (network topology, bandwidth, traffi c matrix, traffi c source types,...), and, even taking into account recent advances in traffi c analysis and generation

	average queue size	inst. queue size
sharp dropping function.	RED	Tail Drop
smooth dropping function.	gentle RED	Gentle RED Inst.

TABLE I

DIFFERENT BUFFER MANAGEMENT ALGORITHMS

We do not examine the performance of individual TCP flows, but we consider the aggregate performance of the aggregate traffic crossing the examined router. There are two reasons to justify this choice:

- The individual performance of TCP flows has been studied in previous papers and our experimental results were not different from the prior publications [12], [14].
- It makes sense to look at the aggregate traffic behavior in a backbone router. One of several possible goals of backbone providers is the maximization of the goodput† of the traffic traversing their network. There is no such evaluation available for RED in the literature.

Note that we do not consider active queue management schemes for traffic differentiation such as RIO or WRED. In these schemes, the dropping function plays a completely different role than in RED. There, it differentiates between traffic classes rather than being (primarily) a congestion control mechanism.

The rest of the paper is organized as follows. In Section II, we first define the metrics we have chosen to analyze AQM mechanisms. Then we describe our evaluation environment which is based on experimentation and simulations. In section III we first compare RED and Tail Drop using the RED parameter values proposed in [5]. In the second part of section III, we observe how varying the averaging parameter influences the performance of RED. In section IV, we show that using a smoother dropping function is not an improvement over RED for TCP traffic but is an improvement over RED performance with regard to consecutive losses. In section V we simulate a new AQM mechanisms that consists of using the smooth drop function of RED with the instantaneous queue size. We discuss this approach (which we call Gentle RED with instantaneous queue size, or GRED-I) and show it is a slight improvement over Tail Drop without the complexity of RED's computation of the average queue length.

II. EVALUATION ENVIRONMENT

Our analysis of AQM is primarily based on experimental results using common, currently deployed routing equipment. Since we used commercial equipment, we were able to vary the RED parameters but could not change the RED implementation. As a consequence we used simulations to complement the experimental results. Our simulations and experimental results have been compared and our simulation has been designed to reproduce the behavior of RED observed in our experimental evaluation.

† In this paper we use the term goodput as the number of bits per unit of time forwarded to the correct destination interface, minus any bits lost or retransmitted.

For both simulations and experimentations, we needed network traffic to load the routers and the network links. We used data provided by a carrier, as well as CAIDA traffic statistics, to generate traffic that is similar to "real" IP backbone traffic with higher number of flows and a high network load ($\rho = 1.1$ to $\rho = 1.5$). The experimental traffic was generated with a scripting tool called Chariot [16].

We also use an analytical model described in [17]. This model helps us to observe or explain behaviors that were difficult or impossible to understand by simulation or experimentation.

A. The metrics

The problem addressed in this paper is not to look at the effect of AQM on each elementary data flow in a router, but to *observe the effects of AQM based on the aggregation of multiple flows* in a router. Consequently, we had to define specific metrics that could describe accurately the aggregate behavior of AQM in routers.

We are not considering per flow performance or fairness amongst TCP flows, although these measures might well be important for service level agreements or customer satisfaction. However, we are interested in how responsive traffic and non-responsive traffic are treated by an AQM scheme.

Thus, the following metrics have been chosen to evaluate the aggregate behavior of AQM:

- *Aggregate goodput of the examined router interface*. This metric reflects the best use of the available router resources. We conjecture that the higher the aggregate goodput, the better the goodput of individual user. We verified this assumption by comparing the goodput of single flows as well as by verifying the goodput of links with different RTTs.
- *The number of consecutive losses*. The combination of TCP's tendency to keep queue occupancy high, of many small HTTP transfers resulting in bursty TCP traffic, and of the Tail Drop bias against bursty traffic, means that loss events at a router tend to involve many packets. If these packets belong to different TCP connections, these connections experience losses at about the same time, decrease their rates/windows in synchrony, and then tend to synchronize. This phenomenon, referred to as the synchronization of multiple TCP connections, has been observed in simulation [18], however it is hard to observe in the operational Internet.

In any case, one claim made by the RED designers is that, since RED spreads out packet drops, it will help break the synchronization pattern which (is thought to) occurs with Tail Drop. To investigate this claim, we examine in this section the impact of RED on the distribution of the number of consecutive packet losses in a loss event at a router.

- *Queueing behavior of the router*. This last metric is only measurable by simulations as it was not possible to collect the required information from the routers in our testbed.

With the queueing behavior we examine the variation of the queue size over time. Variation in the instantaneous queue size is visible as jitter at the end hosts. It degrades performance for (i) TCP flows, as TCP calculates the variance to determine the spacing of the TCP ACKS and (ii) audio/video flows, as those applications install a large playout buffer at the receiver when the jitter is high.

B. The experimental environment

We next describe our test environment, as well as the parameter settings we used during the tests.

The testbed topology (Figure 2) represents a gateway where many networks are merged into one outgoing link and hence create a potential congestion point. Routers used on the testbed are CISCO

7500 routers running IOS 12.0. In this configuration the bottleneck link is between the two routers. At the egress router, no packet will be dropped as the maximum arrival rate is 10MBit/s and the links between the second router and the traffic sinks have sufficient capacity to handle the arriving traffic.

Traffic is generated by high end PCs (Dell Precision 610, Pentium II 450MHz, 380MB RAM) running Windows NT 4.0 and Linux (kernel version 2.2.x). Both operating systems implement TCP NewReno congestion control mechanism. Data sources are the PCs to the left side of figure 2, data sinks are the hosts on the right side of the figure. The network links are Ethernet 10 MBit/s. Between the two routers, we have a full-duplex Ethernet link to ensure that Ethernet collisions do not degrade performance.

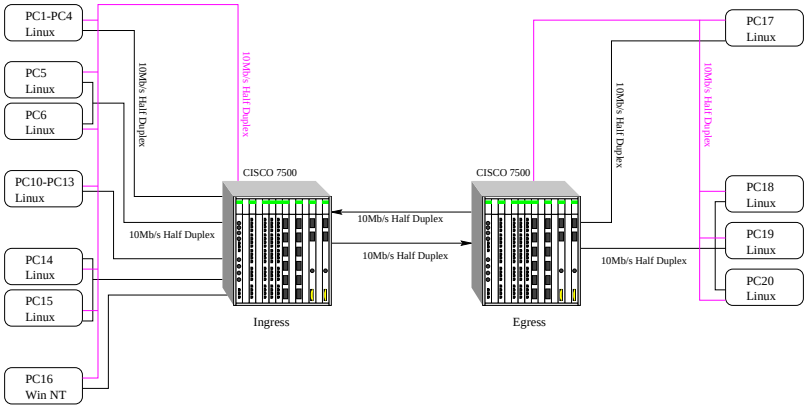

Fig. 2
TESTBED SETUP FOR MEASUREMENTS WITH TWO ROUTERS

We used a custom delay box (made of a PC with two Ethernet cards) so that we could vary the delays on some of the Ethernet links between the sources and the first router. Hence, our testbed allowed us to introduce different transmission delays (varying between 10ms and 200ms) for the different connections [‡].

For all experiments, we used a router buffer size of 200 packets. This corresponds approximately to the bandwidth-delay product of our testbed [§]. For all RED experiments we used the following parameter sets: the minimum threshold $min_{th} = 30$, the maximum threshold $max_{th} = 130$, and the maximum drop probability $max_p = 0.1$. These values are recommended in the CISCO IOS online documentation [19] and are also proposed in [5].

In this paper, "realistic" network traffic means a traffic pattern that can be found on an Internet backbone. Since most of the Internet traffic today is Web traffic, we used sources that initiate small and large file transfers based on the HTTP and FTP protocols. This means we did not use the infinite greedy TCP sources often used in network simulations.

[‡] Note, that we could only vary the delays of a complete link, not of individual flows. Therefore all flows using this specific link suffered the same artificial propagation delay.

[§] Our traffic generator uses TCP packets of 1000 bytes.

The Chariot 2.2 load generator [16] is driven by script files that start and stop data streams on host PCs. Since Chariot scripts use actual application code (from applications such as e-mail, database updates, multimedia conferencing, Web) to generate traffic, we believe that if application parameters are chosen carefully, it provides a method of generating a realistic traffic mix. Unlike simple packet generators, which produce a steady stream of unchanging data, Chariot generates application traffic that is bi-directional, variable, and uses the "real" underlying protocol stack of Windows NT and Linux. Our Chariot scripts did not start all applications at the same time, thus providing an appropriate mix of TCPs in both startup and steady state. Chariot collects statistics during the test runtime but does not report the statistics until the end of the test so as not to perturb the tests with undesired traffic.

We defined the following scenarios (experiments). For the evaluation of RED, we defined 5 experiments (numbered Exp1 to Exp5) where we generated only TCP traffic with respectively 256, 128, 64, 32, and 16 sources. We used these scenarios to evaluate the impact of the number of flows on the end to end performance. For the evaluation with “realistic” Internet traffic we used three additional, different scenarios:

- Experiment Exp6 with 400 sources, with TCP and UDP traffic. Our traffic reflects the current traffic mix in an Internet backbone, namely 10% of UDP traffic, the remaining being TCP (NewReno) traffic [20]. 20% of the TCP traffic is generated by FTP sources sending 100 Kbytes files. The other 80% is HTTP traffic for text (small transfers) and pictures (large transfers).
- Experiment Exp7 with 200 sources and 10% of UDP traffic, TCP traffic as described above.
- Experiment Exp8 with 200 sources and 5% of UDP traffic, TCP traffic as described above

The 10% and 5% of UDP traffic means that we inject this amount of traffic into the network. Due to the dropped packets, this fraction might change after the congested router. We run each experiment for 10 minutes and repeated each experiment at least 10 times. The plotted results correspond to the average of all runs.

C. The simulation environment

The simulation topology is similar to our experimental testbed. All simulations were performed using a modified version of NS 2.0 [21]. 32 source nodes s_i are sending up to 400 TCP and UDP traffic flows to destinations d_i via a RED or Tail Drop router, as shown in Figure 3. The traffic generated is also close to the one we used on the experimental testbed. I.e., many sources on smaller networks compete for the resources of the outgoing link in the ingress router. Note, that in the simulation study, we had to use infinite greedy TCP sources that are only once in the slow start phase since NS does not support TCP connections that restart after the last file transfer is finished, in opposite to the TCP flows in the experimental environment.

The TCP flavor used is the NewReno. The UDP sources send CBR traffic. We use different propagation delays for the links between the sources and the router. The round trip delays vary between 120ms and 220ms. This is different than the experimental testbed, but simulations with too small differences in the RTT tend to generate synchronization effects that could not be observed on our testbed. We therefore used a larger variety of RTTs to minimize the possibility of such an artificial synchronization. In the ingress router, we choose the buffer management scheme to be Tail Drop, RED, GRED and GRED-I. In the egress router we use Tail Drop. However, this is not of interest as the second router is configured so that we do not drop any packets while going through

to. In our simulations, we use more than 200 TCP connections simultaneously sending packets from the sources to the destinations. We also have UDP connections sending at a constant rate which, summed over all UDP sources, equals 10% of the bottleneck link speed. As with our experimental setup, the bottleneck link is between the two routers and has a bandwidth of 10Mb/s. We measure in the first router the drop rates of both UDP and TCP traffic, and the delay of UDP packets. We also compute the aggregate goodput of the TCP connections.

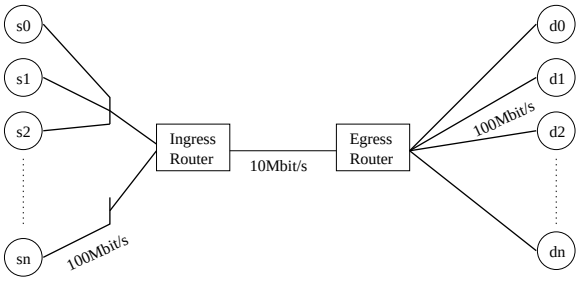

Fig. 3
NETWORK TOPOLOGY FOR THE SIMULATION STUDIES

We run each simulation for more than 200 seconds to minimize the influence of the TCP startup phase (the exponential increase of the sending window). Due to the infinite greedy TCP sources, the 200 second period is long enough to work under stable and comparable conditions for the different tests. Such a setting is clearly different from the "real world" traffic described in the experimental environment. But, as ns did not provide constantly restarting TCP sources, we decided to use the "classic" infinite greedy TCP source to complement the experimental studies.

III. AQM WITH SHARP DROPPING FUNCTION

In this section, we study AQM mechanisms characterized by a sharp dropping function and we vary the queue size averaging. Tail Drop is at one end of the spectrum (sharp drop function and instantaneous queue size) and RED covers the rest of the spectrum (see Table I).

In the first subsection, we use RED parameters proposed in the first revision of RED [5] and we show that the aggregate TCP performance is about the same for RED and for Tail Drop. However, we can show that RED causes UDP loss rates that are higher than the UDP loss rates of Tail Drop. In addition, we observe an increased number of consecutive drops with RED.

In the second part of this section, we vary the value of the averaging parameter in RED in order to evaluate the impact of averaging on RED performance. Other RED parameters are not studied as previous works have shown that varying min_{th}, max_{th} or max_p with a sharp dropping function has a limited influence on the TCP goodput [12], [11].

A. RED vs. Tail Drop

In the first performance comparison, we simply compare RED and Tail Drop performance (using on the 3 metrics defined earlier) in different traffic scenarios. With Experiment 1 to 5, we examined

the influence of the number of flows on the aggregate end to end performance and Experiments 6 to 8 to measure the performance with “real world” traffic.

A.1 Aggregate goodput

Figure 4 plots the realized goodput of all TCP connections with a router running successively RED and Tail Drop.

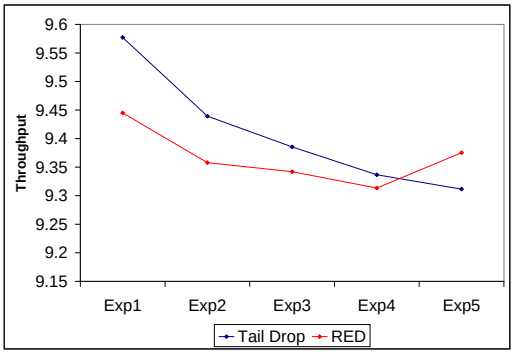

Fig. 4
AGGREGATE GOODPUT WITH RED AND TAIL DROP FOR DIFFERENT TRAFFIC MIXES

We performed different experiments by varying the traffic mix (i.e. number of flows, proportion of TCP/UDP traffic; proportion of long/short transfers, types of applications). For experiment 1 to 5 (shown in Figure 4) we used the same network settings but decreased the number of active flows from 256 to 128, 64,32, and 16 flows. We did not generate UDP traffic in these first experiments. The TCP traffic was dominated by large file transfers (with 80%), the remaining part of the TCP traffic being HTTP with small transfers. We use these scenarios to examine the impact of the number of flows on the router performance and also to be comparable to the simulation studies. In Figure 5 we plot Experiments Exp6 to Exp8 which reflect “real” Internet traffic and uses 400 and 200 sources and different traffic mixes: 80% of the TCP flows is HTTP transfers with small transfer sizes (text and small gifs) and 20% of large files (such as would be generated by large gifs or file downloads). This different traffic patterns reduced the aggregate goodput to about 8.7 MBit/s.

We performed each experiment at least 10 times, with each experiment running for 10 minutes. The goodput plotted is the average of all 10 test runs ¶.

In Figure 4, we observe the relative goodput TCP with RED and TD for different numbers of TCP flows. For more than 32 flows, Tail Drop gives better goodput than RED. For all experiments except Exp5, Tail Drop provides a better goodput than RED. In Exp5, we have only 16 TCP sources.

¶ The standard deviation was smaller than 0.2

For this specific traffic pattern (very few sources), the "random drops" with RED forced individual sources to back off early and allowed a better utilization of the buffer resources. Note, that this result confirms the observation of early RED evaluations where the number of TCP connections were small. [17], [15], [11] explain why RED performs better than Tail Drop when used with few TCP sources. Our experiments with Internet-like traffic support this observation.

A.2 UDP drop probability

Figure 5 compares RED with Tail Drop routers using two metrics: the TCP goodput (as described above), and the loss probability of the UDP flows.

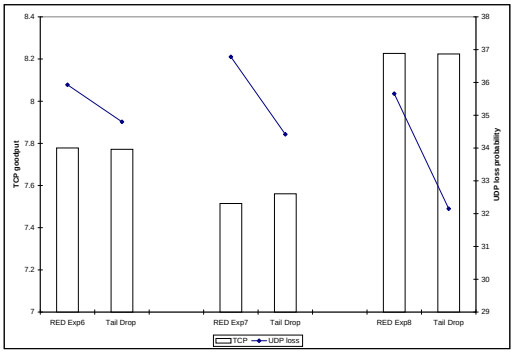

Fig. 5
AGGREGATE TCP PERFORMANCE AND UDP LOSS PROBABILITY WITH RED AND TAIL DROP

Here, we plot the results of the experiments 6 to 8 described above. The plots are again the average of multiple test runs (10 per plot).

We observe that (i), the aggregate TCP performance with RED is comparable with the one realized with Tail Drop and (ii), the drop probability of the UDP flows is significantly higher with RED than with Tail Drop (5% to 12% higher). We conclude that RED does not have a major impact on the TCP performance. The main effect of RED is to increase the drop probability of the UDP traffic. For the Internet this can be interpreted as follows: the aggregate loss rate suffered by TCP connections when going from Tail Drop to RED does not change much, but the loss rate suffered by UDP/IP telephony applications (whether they are rate adaptive or not) will increase significantly. This surprising result illustrates that RED can be harmful towards perfectly behaving sources. Even TCP-friendly traffic sources will suffer more losses compared to TCP sources! This phenomena is also described and explained in [17].

We have consequently been able to show that, in the case of large number of TCP connections and IP-like traffic mix, RED (compared to Tail Drop) increases the drop probability of the UDP traffic,

without improving the goodput of the TCP traffic. The worst impact of RED being to increase the number of consecutive losses.

A.3 Queueing behavior

The queueing behavior has been studied by simulation only. I.e., we used 200 infinite greedy TCP sources with various RTTs and UDP sources sending CBR traffic (10% of the bottleneck bandwidth). The left curve in Figure 6 plots the queueing delay in the router with RED and Tail Drop. The duration of the simulation is 200 seconds and the buffer size is 200 packets. In the figure, we only plot 15 seconds as after the start-up phase the queueing behavior is very stable (due to the infinite greedy TCP sources).

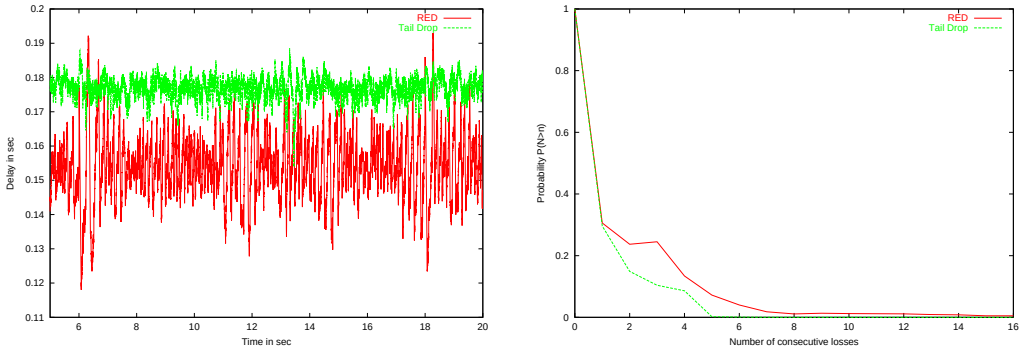

Fig. 6
QUEUEING DELAY AND THE DISTRIBUTION OF THE NUMBER OF CONSECUTIVE LOSSES WITH RED AND TAIL DROP

With Tail Drop, and given the high load in the router, the buffer occupancy quickly increases and then remains close to its maximum value. Note that, with 200 TCP flows and different round trip delays, we do not observe system-wide synchronization patterns that would indicate large scale TCP synchronization. The situation is quite different with RED. The queue builds up quickly; RED starts dropping packets when the average queue size reaches min_{th}, then drops all packets when the average queue size reaches max_{th}. The drop rate decreases when the average queue drops below max_{th}, traffic picks up, the average queue tends toward max_{th} and eventually exceeds it, and the cycle resumes. Thus, as expected, the average queue size stays close to max_{th}, and the RED router behaves essentially like a Tail Drop router with buffer size max_{th} [11], [15]. However, the instantaneous queue size varies heavily with time, more than a Tail Drop queue would do in the same situation. Again, we suspect the sharp dropping function and the averaging function to interact in order to produce the delay oscillation we observe figure 6.

In the next subsection, we will consequently analyze the impact of the queue size averaging functions on the metrics we just evaluated.

A.4 Number of consecutive losses

To evaluate the number of consecutive drops with RED and Tail Drop we used the experimental testbed with 400 sources (Exp6). At the destination we traced the packet arrivals of the UDP flows to determine the number of consecutive losses in their data stream. The plotted values are the average of all UDP flows. We claim that every flow (TCP and UDP) will encounter a similar distribution of the consecutive losses. All our tests did indeed show the same distribution of the number of consecutive losses for the observed flows.

The right plot in Figure 6 shows the probability distribution of the number of consecutive losses for this traffic.

The probability of suffering consecutive losses appears to be significantly higher with RED than with Tail Drop. For example, the probability of loosing more than 5 packets in a row is one order of magnitude higher with RED than with Tail Drop. While we measured more than 15 consecutive losses in *one individual* flow with RED, we never observed more than 7 consecutive drops with Tail Drop. Note that this plot only shows the probability distribution of each individual flow. Of course, we find higher probabilities of consecutive losses when we examine the loss process of all aggregate flows (as plotted in Figures 9 and 11). Earlier work in a Tail Drop environment showed that the two measures can be very different from each other indeed [22].

This is a very important result that we attribute to the interaction of the sharp edge of the dropping function with the averaged queue size of RED. When the load at the gateway is high (larger than 1[‖]), the average queue size of RED will be close to the value of max_{th}. When the average is larger than max_{th} RED will drop all incoming packets until the average decreases below max_{th}. Due to the slow reaction time incurred by the averaging, a RED gateway drops more packets than a Tail Drop gateway.

We conclude, that the average function in coexistence with the sharp dropping function will always perform badly in terms of the number of consecutive losses.

B. Impact of averaging on RED performance

We conjectured in the previous section that the queue size averaging would be likely to explain some of the RED pathologies that are not observed in Tail Drop. In this section, we study how RED's performance vary with the averaging function. We believe the averaging to be a key parameter in the efficiency of RED. If the average queue size "reduces" the reactivity of RED to traffic changes, on the other hand, it bases the computation of the drop rate on a completely artificial value that does not reflect the real amount of data in the queue. Because the average queue size can be far from the real queue size, it is important to understand the influence of the averaging on the RED performance. We illustrate with two example situations where averaging can be critical. It is well known that carriers over-engineer their backbone, the rule being not to use more than 50 to 60% of the link bandwidth to be able to absorb the sudden traffic growth in case of link failure. Here, using the average queue size instead of the instantaneous queue size could be harmful:

- In normal conditions, we don't want to lose any packets when the queue is used for less than 50% of the available bandwidth. When one link fails, the operational link will suddenly double its load, and it will take some time to the averaging to reflect this change. If the increase due to a failure of

[‖] Note, that we measured a router load larger than one during our experiments, even in the present context of TCP traffi c!

a link is 20%, the carrier does not want to lose any packets. Failure recovery is expected to be fast enough so that the traffic will come back to normal in few seconds.

- After a peak of traffic (when the broken link is restored), the average queue size will stay high even if the queue size is back to normal. Hence, RED will continue dropping packets even after the congestion.

This was indeed the intention of the inventors of RED to take advantage of the averaging and they defined a parameter to modify the smoothing factor in the averaging function. However, they did not explain how to find the best value for this parameter. If w is too large, then the averaging procedure will not filter out transient congestion at the router and does not offer the buffer space to newly arriving bursts. That is when RED behaves like a Tail Drop router. When w is set too low, then RED responds too slowly to changes in the actual queue size and packets are dropped due to a buffer overflow. When a burst arrive at the router and even if a buffer overflow can be avoided, RED will still drop packets with a higher probability after the congestion is gone. Hence, the average queue size is helpful to filter out a congestion, but is harmful when the congestion is gone.

We now examine the influence of the averaging parameter on TCP performance and UDP loss probability. Figure 7 plots the TCP goodput and the UDP loss probability as a function of an increasing value of w. We use the simulation setup described in section II.

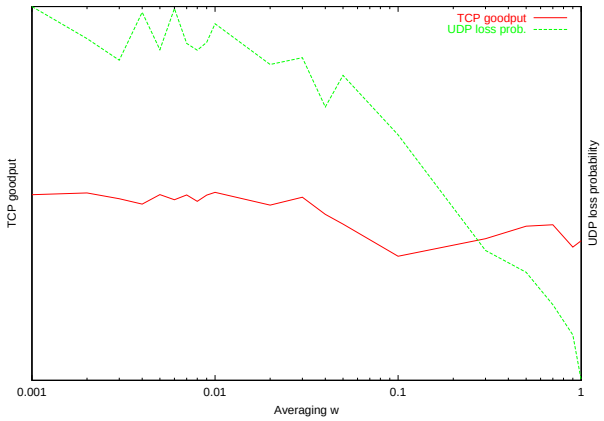

Fig. 7

IMPACT OF THE AVERAGING PARAMETER w ON THE AGGREGATE TCP PERFORMANCE AND THE UDP LOSS PROBABILITY

We observe that the TCP goodput is not very sensitive to the value of w. However, the UDP loss probability is decreasing when w is getting larger. The best UDP performance is realized when the instantaneous queue size is used. This result confirms our observation in the previous section which is that RED penalizes more the UDP traffic than the TCP traffic. We can now add that this is due to the averaging of the queue size with the sharp edge in the dropping function.

C. Summary of findings

We now summarize the findings of this section. We found that RED with the average queue size and the sharp edge in the dropping function:

- does not increase the goodput of the router interface,
- does increase the drop probability of the UDP flows,
- adds heavy variations to the actual queue size and consequently add jitter in the data path, and
- creates a higher number of consecutive losses.

The last point is very important and can be explained by the interaction between the sharp edge in the dropping function and the average queue size. When the router load is high, the average queue size increases and start oscillating around the value of max_{th}. Whenever the average queue size is above max_{th}, RED will drop all incoming packets until the losses forces the sources to back-off. Therefore, we encounter a high number of consecutive losses.

We now investigate modifications to the RED algorithm that should help to solve the above problem. One approach is to use the average queue size as the congestion indicator and to avoid the sharp edge in the dropping function (gentle RED [6]). The other alternative is to use the instantaneous queue size instead of the average the queue size. Note that, when the instantaneous queue size is used, there is no obvious reason for choosing max_{th} smaller than the buffer size and $max_p < 1$. We called this approach gentle RED with instantaneous queue size (GRED-I).

IV. RED VS GENTLE RED

Recently, based on the findings in [23], the RED algorithm has been modified with a new dropping function. In this section we evaluate the performance of this modification to RED. We now analyze the left column of the AQM design space (Table I) by comparing RED and gentle RED.

Gentle RED represents the most recent modification to RED. Basically, the new dropping function avoids the sharp edge we suspected to be harmful in the classic RED definition. Instead of the step function at max_{th}, Gentle RED slowly increases the drop probability between max_{th} and $2 * max_{th}$ from max_p to 1. The GRED dropping function consequently looks closer to an exponential function than to a step function. Gentle RED is not yet supported by router's vendors. In order to study GRED on our experimental testbed, we approximated it by simply using a maximum drop probability max_p of 1 and a higher value for the max_{th}. In our experiments we set max_{th} to 130 packets, where the maximum buffer size was set to 200 packets.

To compare RED with gentle RED, we used (such as in figure 5) the three experiments with 200 and 400 sources described in II, i.e., Exp6, Exp7, and Exp8. Again, we run each test 10 times and plot the average of the 10 test runs [**].

In Figure 8, we compare the results of these tests in terms of the aggregate TCP goodput and the UDP loss rate. The first conclusion we can draw from these experiments is that RED and GRED perform differently. We observe first that the UDP loss probability with GRED is significantly lower than with RED (about 10% lower). Figure 8 also shows that the TCP performance is better with RED than with GRED (about 3% higher with RED). Even though, the results show only little impact on the performance of the TCP traffic.

[**] Here, the standard deviation was smaller than 0.1

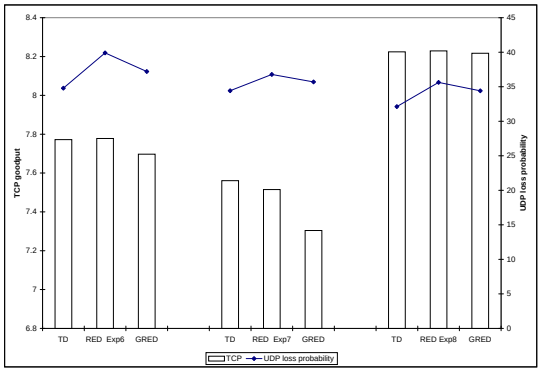

Fig. 8
AGGREGATE TCP PERFORMANCE AND UDP LOSS PROBABILITY WITH TAIL DROP, RED, AND GENTLE RED

Figure 9 plots the simulation and analytical results of the two missing metrics, i.e., the queueing behavior and the number of consecutive losses.

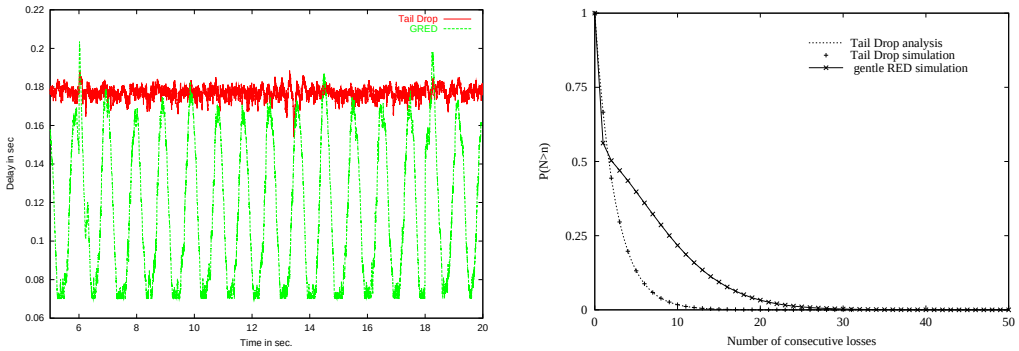

Fig. 9
QUEUEING BEHAVIOR AND THE DISTRIBUTION OF THE NUMBER OF CONSECUTIVE LOSSES WITH GENTLE RED AND TAIL DROP

Figure 9 left plots the simulation results of the evolution of the actual queue size over time. We observe heavy oscillation of the actual queue size with GRED. The averaged queue size with the "gentle" RED function leads obviously to a synchronization of the TCP flows. Note, that we used connections with different RTTs and yet observe this strong phase effect!

Figure 9 right plots the distribution of the number of consecutive losses with gentle RED and Tail Drop. We used the model described in [17] to simulate a GRED router for an offered load of $\rho = 2$. The results show that GRED increases the probability to encounter multiple drops. This indeed can explain the oscillating behavior of the GRED queue.

We conclude that our investigation on the performance of gentle RED on an aggregation of flows shows that the new dropping function with the maximum drop probability of 1 does not improve the end to end performance compared to Tail Drop and RED.

V. GENTLE RED WITH INSTANTANEOUS QUEUE

We have seen in section III that a sharp drop function in cooperation with an average queue size can reduce the performance of a flow aggregation. In section IV, we examined an evolution of RED which consisted in changing the sharp dropping function in a more regular (exponential) drop function. In this section we evaluate the last part of the AQM design space (as defined table 1), i.e. drop packets with a gentle RED-like function, based on the instantaneous size of the queue.

As we have seen before, there are many reasons to prefer a dropping mechanism based on the instantaneous queue size:

1. Averaging can be harmful since it takes time to recover from congestion period. While the higher drop probability is useful during the congestion, it is harmful whenever the load of the router is decreasing.
2. Averaging helps to avoid the bias against bursty flows. However, it is questionable if this is a desirable goal in a real network (the bursty nature of arrivals might not require an increase of the loss rate)
3. Averaging does not help to anticipate congestion. There must be better congestion indicators than the average queue size. In particular, the average queue size does not reflect the real state of the queue.

In the remaining of this paper, we investigate the last quarter of our AQM design space, i.e. gentle RED with no averaging (GRED-I). We analyze GRED-I with the value of the metrics defined above. Results shown in this section are not obtained with experimentations but with simulation and the analytical model we borrowed from [17]. Here, we used the following settings: We used a minimum threshold $min_{th} = 30$, but set the maximum threshold max_{th} to 200 packets, i.e., the maximum buffer size. There is no reason to use a smaller maximum threshold since we want to maximize the use of the allocated resources.

In Figure 10 we plot the TCP goodput of all TCP connections with GRED and GRED-I. The figure shows the three different experiments as described earlier. Exp6 was realized with 400 TCP sources sending and 10% of UDP/CBR traffic and the setup described in section II. The results of Exp7 were obtained with 200 and the ones of Exp8 with 200 TCP sources but only 5% of UDP traffic.

As illustrated by figure 10, GRED-I increases the TCP goodput and decreases the dropping probability of the UDP flows compared to GRED. Clearly, GRED-I outperforms GRED. In terms of TCP goodput, GRED-I performs best; only Tail Drop offers better results for the UDP drop probability. Note, that we obtained these results with simulation (i.e., we had to use infinite greedy TCP sources instead of multiple file transfers) and hence, the results differ from the ones obtained on the testbed!

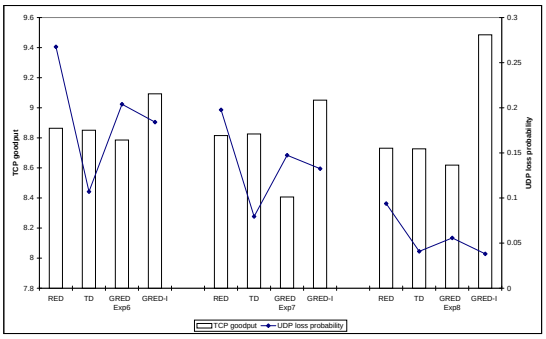

Fig. 10

AGGREGATE TCP PERFORMANCE AND UDP LOSS PROBABILITY WITH TAIL DROP, RED, GRED, AND GRED-I

Figure 11 plots the simulation results of the last two metrics, i.e., the queueing behavior and the number of consecutive losses.

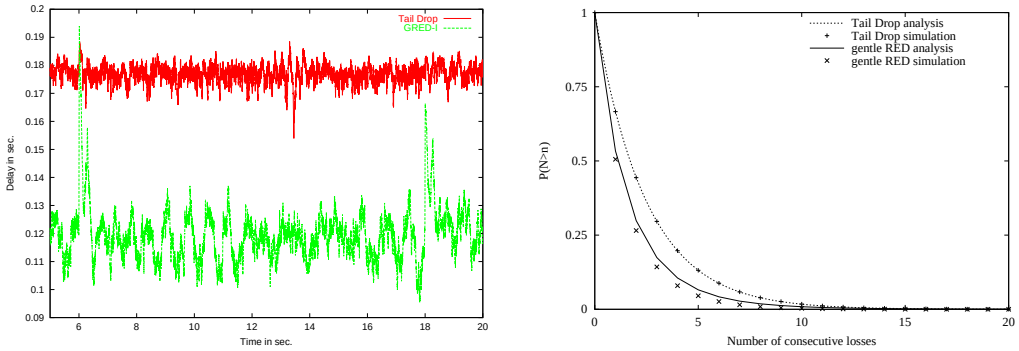

Fig. 11

QUEUEING BEHAVIOR AND THE DISTRIBUTION OF THE NUMBER OF CONSECUTIVE LOSSES WITH GENTLE RED-I AND TAIL DROP

These results clearly show the advantage of the mechanisms that do not rely on the average queue size. The use of the instantaneous queue size reduces the oscillations of the queue size. Moreover, the average delay is 50% shorter with GRED-I than with TD††. In addition, the use of a smooth dropping function with the instantaneous queue size can reduce the queueing.

†† Note that this percentage is very sensitive to the buffer size and can signifi cantly vary with it

Figure 11 right shows the number of consecutive losses with GRED-I and Tail Drop. While the number of consecutive losses is still higher with GRED-I, than with TD we can see a dramatic improvement compared to the results of GRED (compare with figure 9).

We conclude that GRED-I has got some very interesting properties compared to Tail Drop, like the reduced delay and the improved TCP goodput. However, there is still the problem of a higher drop probability for the UDP traffic and a sightly higher number of consecutive losses. Hence, our investigation of GRED-I can not be considered exhaustive and we plan to experiment and investigate further in this promising variant of AQM.

VI. Conclusion

In this paper we have examined the impact of two elements of the active queue management schemes on the aggregate performance of multiple TCP and UDP flows crossing a router interface. We started by the comparison the upper horizontal line of table I, i.e., RED vs. Tail Drop. For large numbers of flows there is obviously no advantage in using RED instead of Tail Drop. We have shown that many of the RED problems arose from the the interaction between a sharp edge in the dropping function and a queue size averaging. To avoid the problem of multiple consecutive drops, we then examined two solutions to this problem: i) to avoid the sharp edge, and ii) to not use the average queue size. For the evaluation of the first alternative, we compared the vertical elements of table 1, i.e., RED and GRED. We found that GRED is definitely not an improvement compared to RED (RED showed better results for all the metrics).

Finally we examined the last mechanisms in the AQM design space we have proposed, gentle RED with instantaneous queue size. While in our simulations GRED-I can improve the TCP goodput and also decreases the queueing delay, GRED-I produces yet more consecutive losses than Tail Drop but exhibits an increased drop probability for the UDP flows.

Our tests and experimentations have shown that if an ISP or a carrier want to use something different than Tail Drop, gentle RED with the instantaneous queue size is the system that exhibited the best performance. In addition, GRED-I offers a better and faster reaction to congestion and should be easier to implement than the schemes relying on the average queue size.

However, the question of whether the gain from GRED-I is worth the effort required to deploy it in a networks has to be discussed. The gain from GRED-I is small. Moreover, none of the AQM mechanisms (except Tail Drop) is smart enough to solve efficiently the carrier's problem with regards to link failure. Furthermore, we believe that to really offer a better service in a network, the ISPs have to consider a metric we did not evaluate: fairness. We do not believe that any AQM scheme can efficiently match the requirements of a fair network. AQM also does not provide flow isolation and protection from misbehaving flows.

References

[1] V. Jacobson, Congestion avoidance and control, in *Proc. ACM Sigcomm'88*, 1988.

[2] A. Demers, S. Keshav, , and S. Shenker, Analysis and simulation of a fair queueing algorithm, SIGCOMM Symposium on Communications Architectures and Protocols , October 1989.

[3] R. Braden, D. Clark, and S. Shenker, Recommendations on Queue Management and Congestion Avoidance in the Internet in the Internet, RFC 2309, 1998.

[4] S. Floyd and V. Jacobson, Random Early Detection Gateways for Congestion Avoidance, IEEE/ACM Transaction on Networking **1**(4), 397–413, August 1993.

[5] S. Floyd and K. Fall, Router Mechanisms to Support End-to-End Congestion Control, Technical report, Network Research Group at LBNL, 1997.
[6] S. Floyd and K. Fall, Promoting the use of end-to-end congestion control in the Internet, IEEE/ACM Trans. Networking , 1999.
[7] S. Doran, Interface Graphs of a RED-enabled router, http://adm.ebone.net/šmd/red-1.html, 1998.
[8] W. chang Feng, The Impact of Active Queue Management on Multimedia Congestion Control, Proceedings of IC3N , October 1998.
[9] D. Clark, Explicit Allocation of Best Effort Packet Delivery Service, Technical report, MIT Laboratoty for Computer Science, 1997.
[10] W. chang Feng, D. D. Kandlur, D. Saha, and K. G. Shin, Understanding TCP Dynamics in an Integrated Services Internet, in *Proceedings of NOSSDAV*, 1997.
[11] W. chang Feng, D. D. Kandlur, D. Saha, and K. G. Shin, BLUE: A New Class of Active Queue Management Algorithms, Technical report, Department of EECS zNetwork Systems Department University of Michigan, 1999.
[12] R. Morris, TCP Behavior with Many Flows, in *Proceedings of IEEE/ICNP'97*, October 1997.
[13] T. Ziegler, S. Fdida, and U. Hofmann, Stability Criteria for RED with bulk-data TCP Traffi c, Technical Report RP-LIP6-99-14, Universite' Pierre et Marie Curie, Laboratoire Paris 6, August 1999.
[14] D. Lin and R. Morris, Dynamics of Random Early Detection, in *Proceedings of SIGCOMM'97*, 1997.
[15] T. J. Ott, T. Lakshman, and L. Wong, SRED: Stabilized RED, in *Proceedings of Infocom'99*, Infocom, 1999.
[16] Ganymede-Software, Chariot 2.2, http://www.ganymedesoftware.com/html/chariot.htm, 1998.
[17] M. May, T. Bonald, and J. Bolot, Analytic Evaluation of RED Performance, in *Proc. of Infocom'2000*, March 2000.
[18] L. Zhang, VirtualClock: A New Traffi c Control Algorithm for Packet Switching Networks, in *SIGCOMM Symposium on Communications Architectures and Protocols*, edited by ACM, 1990.
[19] CISCO-Systems, IOS Confi guration Guide, http://www.cisco.com/, 1998.
[20] K. Claffy, Internet Measurements, http://www.caida.org/Presentations/Soa9905/, May 1999.
[21] S. Floyd, NS network simulator, http://www-mash.cs.berkeley.edu/ns/, 1995.
[22] J. Bolot, S. Fosse-Parisis, and D. Towsley, Adaptive FEC-Based Error Control for Interactive Audio in the Internet, in *Proc. IEEE Infocom '99*, March 1999.
[23] V. Rosolen, O.Bonaventure, and G. Leduc, A RED discard strategy for ATM networks and its performance evaluation with TCP/IP traffi c, Computer Communication Review **29**(3), July 1999.

CONTENTS

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399