TUF : Tag-based Unified Fairness

Antoine Clerget, Walid Dabbous

Abstract—**Finding an appropriate end-to-end congestion control scheme for each type of flow, such as real-time or multicast flows, may be difficult. But it becomes even more complex to have these schemes be friendly among themselves and with TCP. The assistance of routers within the network for fair bandwidth sharing among the flows is therefore helpful. However, most of the existing mechanisms that provide this fair sharing imply complex buffer management and maintaining flow state in the routers. In this paper, we propose to realize this fair bandwidth sharing *without per-flow state* in the routers, using only a trivial queueing discipline. Packets are tagged near the source, depending on the nature of the flow. In the core of the network, routers use FIFO queues, and simply drop the packet with the highest tag value in case of congestion. Contrarily to other stateless fair queueing algorithms in the core routers, we do not try to maintain instantaneous flow rates equal. Instead, we take into account *the responsiveness nature* of the flows, and adjust loss rates such that average rates are equal. The novel approach of our scheme, called TUF , Tag-Based Unified Fairness, not only improves the overall fairness but enables us to maintain it in *realistic environments*, with non-negligible round trip times or bursty traffic, where other schemes fail. The corresponding cost is the need for models of the end-to-end responsive natures of the flows.**

Keywords—**Stateless fair queueing, end-to-end congestion control, multicast, responsive flows, TCP, max-min fairness.**

I. INTRODUCTION

One of the challenges in the design of modern networking applications today is the efficient and cooperative use of network resources. With the increasing greediness of multimedia applications over the Internet, one must be concerned both with fairness with other applications and adapting the sending rate to the receiver's capacities - in other words with congestion and flow control. There are basically three ways to consider how an application should determine its sending rate :

- Send at a fixed rate.
- Estimate network conditions and adapt accordingly in the end hosts, either through a specific protocol (TCP) or by the applications (adaptive applications). This is the end-to-end approach.
- Rely on mechanisms in the routers to ensure fairness. This is the network approach.

The first approach, unfortunately adopted by most multimedia applications today, is not satisfactory, as it is not efficient and fair. In the best case, these propose the user to select among a set of predefined rates, such as for audio applications, LPC, GSM, ADPCM. The two other approaches are discussed below.

A. Relaxing the end-to-end dogma

To share network resources among competing flows, we have so far mostly relied on pure end-to-end mechanisms, in particular TCP. Fairness is achieved by congestion avoidance schemes in the end hosts. By reacting to aggregate feedback from the network, usually in an additive increase - multiplicative decrease manner, we reach a "fair" allocation of the critical resource, i.e. the bottleneck bandwidth [1]. TCP sources sharing a common bottleneck tend to synchronize because they undergo the same loss rate.

However, the success of this approach, with the widespread use of TCP, relies on the following assumptions :

- *Applications all use a TCP-friendly adaptation scheme.*

To ensure fairness, all flows should use a TCP-friendly adaptation scheme [2]. With additive increase, multiplicative decrease (AIMD) schemes, resources are not necessarily evenly shared if the AIMD parameters are not chosen equal in the end hosts. Using an AIMD adaptation scheme for all types of flows, such as multimedia flows with real-time constraints and data transfers with bandwidth requirements, is not appropriate [3]. Hence, equation-based congestion control protocols, such as TCP-Friendly Rate Control (TFRC) [4] have been proposed to change rates more smoothly. However, these adapt more slowly to transient changes in congestion. Their implementation for all kinds of flows, such as audio flows, may not be convenient. We believe it is more realistic to consider that we will always be confronted to the presence of flows with varying agressivity or responsiveness.

- *A same aggregate feedback from the network enables the TCP-friendly flows to converge*

The upcoming of new media, such as dynamic mobile networks, and new types of flows, such as multicast flows, causes this assumption to fail. In the case of *multicast flows*, our source is sharing multiple bottlenecks with different flows at the same time, making rate adaptation non-trivial. Should we adapt to the slowest receiver ? Finding this TCP-equivalent rate is not as simple as just finding the largest round trip time and the worst loss rate or equivalent window size [5], [6]. Moreover, the interaction of adaptive multicast flows, sender-driven or receiver-driven [7], among themselves or with other unicast flows is not only non-trivial but maybe somewhat hazardous. In the case of *dynamic mobile networks*, such as ad-hoc networks or satellite constellations, there is no fixed bottleneck and the set of competing flows changes due to route instability. In this case, not only is the feedback (network conditions estimate) difficult to obtain and unstable, but so are the real conditions.

Proposing some help from the network and relaxing the end-to-end dogma will prove to be more and more interesting if not necessary with the advent of new links and media in the Internet, and with the increase of heterogeneity among the hosts in the network.

B. The network approach

Using network mechanisms such as fair queueing has the advantage of protecting flows from other aggressive or ill-behaved flows. Most of these fair queueing algorithms such as Stochastic Fair Queueing (SFQ) [8], [9], [10] share the bandwidth between the flows by maintaining a queue per flow. Among these, Deficit Round Robin (DRR) [11] achieves nearly perfect fairness with limited complexity.

However, maintaining that flow state has a cost and raises scalability issues in high speed networks that can jeopardize the massive deployment of these techniques. The success of the Internet is partly due to the fact that most of the intelligence

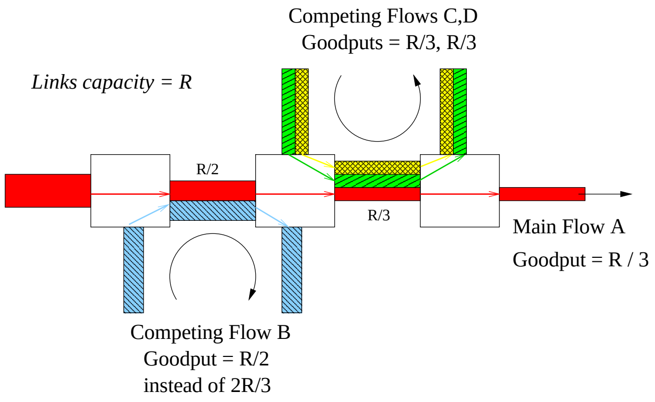

Fig. 1. Inefficient use of the network

was kept in the boundaries of the network and only light mechanisms were introduced in the core of the network. Moreover, dealing with individual flows while routing aggregate flows, for example within IP tunnels where packets are encapsulated, is non trivial. Recently, Stoica et al. proposed *Core*-Stateless Fair Queueing (CSFQ) [12], in order to approximate fair queueing while limiting flow state to the edge of network and removing it from core routers. Cao et al. proposed Rainbow Fair Queueing (RFQ) [13], a similar scheme that avoids fair share rate calculation in the core routers and that is better adapted to layered encoding applications. Both schemes remove flow state but still require computation to determine dropping thresholds for example. CHOKe (CHOose and Keep) [14] defines mechanisms as simple as Random Early Discard (RED) [15] in the core routers. However, it improves but doesn't solve the fairness issue.

C. A hybrid approach

Relying exclusively on the network leads to inefficiency, since the network will use resources to transport packets eventually dropped. In figure 1, the main flow A shares a first link with flow B and a second link with flows C and D. Using exclusively Fair Queuing mechanisms will lead to an unfairness for flow B that could obtain $2R/3$ instead of $R/2$ if the main flow A adopted an end-to-end congestion control. It is thus preferable to combine network mechanisms with end-to-end congestion control.

Existing network mechanisms ignore the end-to-end behaviors of the flows they try to regulate, and consider these flows to be unresponsive. A first class of algorithms, in particular all stateless fair queueing algorithms, try to keep instantaneous rates equal. They are therefore unfair to responsive flows that use less than their fair share rate when reacting to losses (see section II-B). A second class of algorithms focus on average rates, but then need to maintain flow state.

To address these issues, this paper proposes a new stateless fair queueing mechanism. The strength of our approach relies on the following points :

- The mechanisms introduced in the core routers are very **light and introduce no flow state**. Compared to the other approaches that achieve reasonable fairness, we believe that our scheme has one of the lowest implementation cost in the core routers.
- Contrarily to other stateless fair queueing algorithms, we do not consider all the flows to be unresponsive CBR flows, and try maintain *instantaneous* rates equal. Instead, we take into account **the responsive nature** of the flows (e.g. TCP friendliness) and maintain loss rates such that *average* rates remain equal. This novel approach makes it possible to maintain fairness in heterogenous environments with various adaptation schemes.

The paper is structured as follows : in section II we describe our fair bandwidth sharing mechanism, named TUF (Tag-based Unified Fairness). Section III describes implementation and deployment issues. Section IV presents simulations results, comparing TUF to other fair queueing mechanisms. Section V describes our current Linux implementation and experimentations, and section VI concludes the paper.

II. TUF : Tag-based Unified Fairness

A. Objectives

As with Fair Queueing [8] [9] [10], our primary objective is to allocate network resources fairly among the flows in a congested core router. Let's suppose that we have n flows, $F_1, \dots, F_n$. Let $\hat{R}_i$ be the rate of flow F_i upstream (arriving at the router), and R_i be the rate of the flow downstream (leaving the router). The bandwidth allocation is fair if $\forall i, R_i = min(\hat{R}_i, R_{fair})$, where R_{fair} is the fair share rate at the router, i.e. the maximum bandwidth a flow can get. R_{fair} is such that it fills the router's capacity C ; it is the unique solution to :

$$C = \sum_i R_i = \sum_i min(\hat{R}_i, R_{fair})$$

A flow F_i is either :

- Constrained : the bandwidth downstream is, as for all constrained flows, equal to the fair share rate, and is larger than that obtained by unconstrained flows. $\forall j, F_j$ constrained, $R_i = R_j = R_{fair}$
- Unconstrained : the bandwidth upstream is less than the fair share rate. The router therefore forwards all its packets and the local loss rate is 0. $\forall j, F_j$ constrained, $R_i = \hat{R}_i \leq R_j = R_{fair}$

Combined with end-to-end congestion control, we can achieve our fairness objective : global **max-min fairness** [16]. This qualitative property can be summarized as follows : *we say that we allocate bandwidth max-min fairly if it is not possible to increase the satisfaction of a flow without simultaneously causing the decrease in the satisfaction of a less satisfied flow.* The satisfaction of a flow is here measured in terms of average bandwidth.

B. Adjusting instantaneous rates and fairness

We are concerned with fairness between elastic flows for which the satisfaction is measured in terms of average rate. Constraining the flows' instantaneous rates will not lead to fairly distributed average rates. Indeed, when reacting to congestion signals, responsive flows reduce their rate and do not use at all time their fair share. We call this first cause of unfairness the *variation effect*. This is depicted on figure 2, where the bandwidth is shared between a responsive flow such as a TCP-Reno flow, and a very greedy CBR flow.

The performance of responsive flows such as TCP flows can be drastically worsened by the very sudden increase in the loss

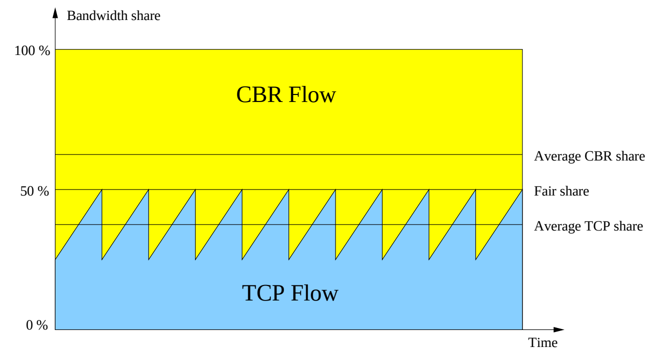

Fig. 2. The variation effet

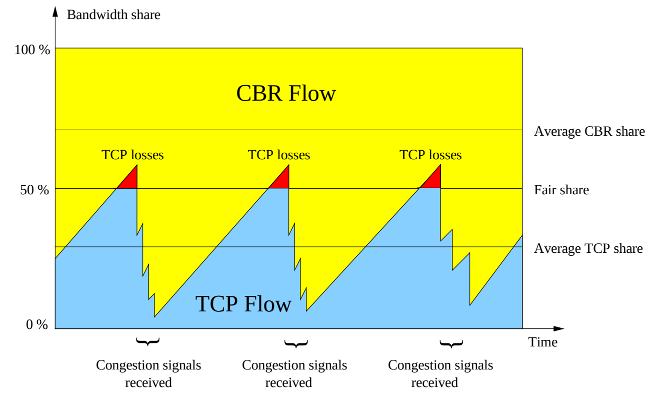

Fig. 3. The burst effect

rate as the flow's rate exceeds the fair share rate. We call this second cause of unfairness the *burst effect*. When the round trip time becomes non-negligible, TCP flows take time to reduce their rate, and will therefore undergo bursts of losses. These bursts also occur after a sudden decrease in the fair share rate, due for example to the sudden arrival of new flows. The *burst effect* will have a serious and durable effect on the average rate as depicted on figure 3.

In TUF , we avoid adjusting instantaneous rates, and adjust loss rates, taking into account the end-to-end behavior of the protocol.

C. Overview

To provide fair bandwidth allocation between flows sending at different rates, we differentiate their loss rates. As no state is present in the routers, state has to be present in the packets to enable distinct behaviors (i.e. loss probability) between the flows. That state is a "Tag", a numeric value carried in one of the packet's field. The congested core router only uses the tags of the packets present in its queue to make a drop decision. A tagging entity, called the "tagger" is responsible for placing a tag in each packet (comparable to the DS-codepoint in diffserv). This entity, that maintains flow state, is either a router at the edge of the network, or ideally the source itself. Tagging should not depend on the state of the network : we do not want specific feedback from the routers in the network. Figure 4 presents our global architecture.

In section II-D, we define the core router's queueing discipline, and in section II-E the tagging algorithm.

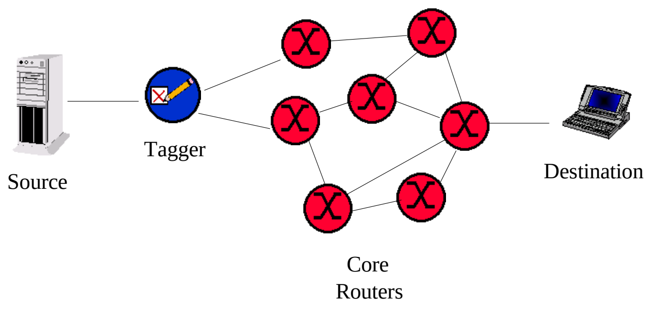

Fig. 4. Our tagging architecture

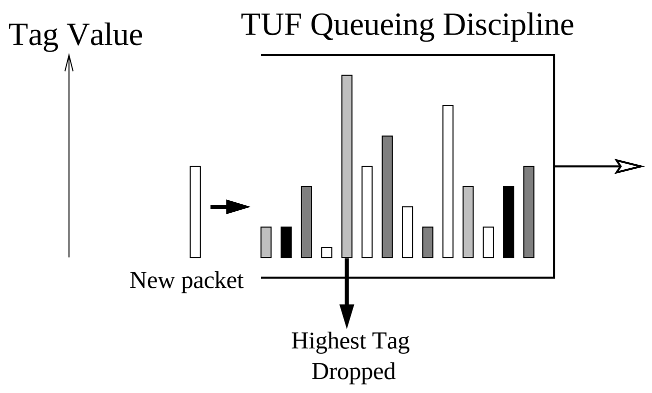

Fig. 5. The queueing discipline in a TUF router

D. Core Routers : The TUF queueing discipline

The Tag is a numeric value carried in a field, in the packet header. It is used by the routers to decide if they should enqueue or drop a packet. We chose to give high values to the tags of the packets that are more eligible for discard, and low values to the tags of packets that are less eligible for discard. The tag therefore represents a **drop precedence**. As FIFO queues, our queueing discipline preserves packet order to avoid negative impact on protocols such as TCP. In case of congestion, i.e. when a packet arrives and the queue is full, the router drops the packet with the highest tag (figure 5). Note that for a same incoming traffic, this queueing discipline will give the same overall loss rate and carried load than the FIFO queue. We here propose a model of this queueing discipline behavior.

The TUF filter model

In the simulation of the TUF queueing discipline (figure 6), the TUF queue is fed with a uniform tag distribution between 0 and 1, an offered load of $\rho = 2$, and poisson arrival and service times. As the queue size increases, the TUF queueing discipline approximates the behavior of a low-tag filter. The packets with the lowest tags, i.e. with tag values below a threshold K, are forwarded, and the others are dropped. The threshold K is such that it fills the router's capacity. Due to lack of space, the formal proof is not included here, but is available upon request. We now consider that the queue size is large enough (e.g. 64 packets) for the TUF filter model to be a good approximation of the router's behavior. The complexity of the search grows linearly with the queue size. We can however limit the search to a high enough number of packets at the head of the queue (e.g. 128) while still approximating the behavior of a low-tag filter.

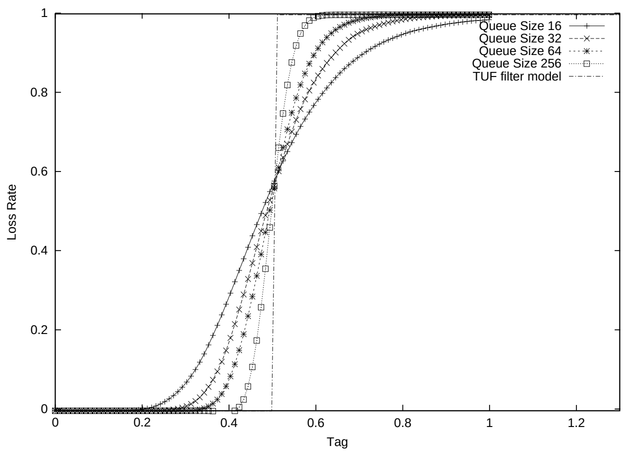

Fig. 6. Tag loss probabilities for the M/M/1/N $\mathcal{T}$UF queue

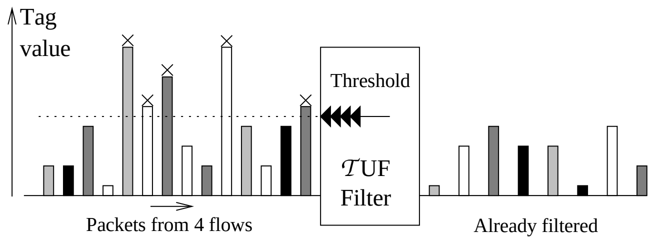

Fig. 7. A $\mathcal{T}$UF router, modeled as a simple filter

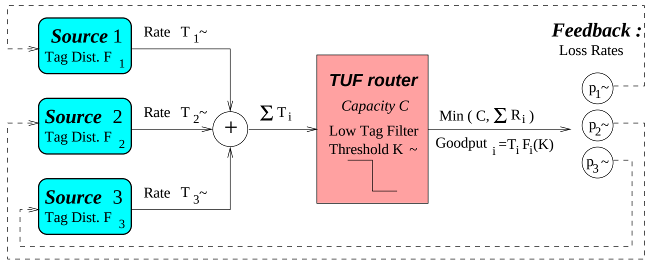

Fig. 8. Modeling end-to-end adaptation schemes

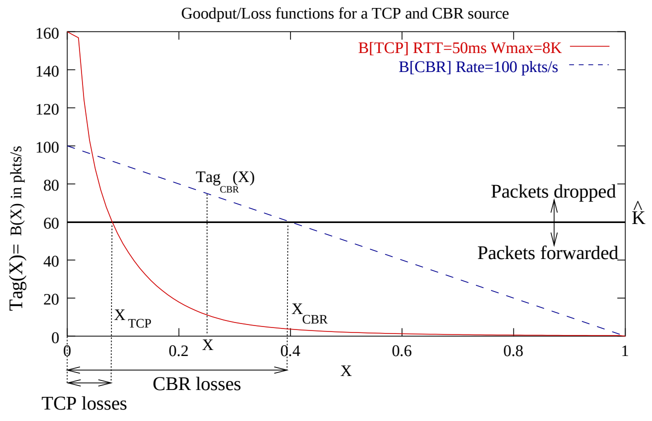

Fig. 9. Tags for TCP and UDP flows

E. *Taggers : The tagging algorithm*

This section describes how packets are tagged to achieve fair queueing in $\mathcal{T}$UF routers, modeled as low-tag filters. One of our objectives is to enable the coexistence of different end-to-end adaptation schemes. We start by proposing a model for congestion control algorithms. We then describe our tagging algorithm and prove that we converge towards a fair state.

E.1 Modeling end-to-end congestion control algorithms

We consider end-to-end congestion control algorithms that adapt the sending rate based on loss rate feedback. We suppose that we know the average goodput achieved by our sources $S_1, \ldots S_n$ when submitted to a Bernouilli loss rate with an intensity p : $B_1(p), \ldots, B_n(p)$. The functions $B_i(p)$ are strictly decreasing continous functions, and $B_i(0) = R_i, B_i(1) = 0$, where R_i is the maximum sending rate. When submitted to a constant loss rate intensity p, the goodput converges towards $B(p)$, and the throughput towards $B(p)/(1-p)$. For the TCP-friendly adaptation scheme, $B(p) = (1-p).Min(W_{max}/RTT, c/RTT\sqrt{p})$.

Figure 8 shows the behavior of the global system. The dynamics of the system is modeled as follows :

- Sources send at rate $T_i(t)$, bounded by R_i.
- Packets are tagged following the cumulative distribution F_i : $F_i(x) = Probability(Tag \leq x)$.
- If the router is congested, the highest tags (above $K(t)$) are dropped, filling the router's capacity.
- Sources receive feedback in terms of packet loss

$$p_i = 1 - F_i(K(t)) \tag{1}$$

- Sources raise or reduce their sending rate accordingly. We suppose that they try to converge exponentially towards their instantaneous target rate $T_i^* = B_i(p_i)/(1-p_i) : (\alpha > 0)$

$$\frac{dT_i}{dt} = \alpha.(T_i^* - T_i) \tag{2}$$

E.2 The tagging strategy

Theorem 1: Let X be a stochastic variable, uniformely distributed between 0 and 1. Given a set of sources S_i, modeled by the loss-goodput functions $B_i(p)$, tagging packets from source S_i with $Tag_i = B_i(X)$ leads to a **stable and fair bandwidth sharing** between the flows.

The tagger proceeds as follows :

- When a packet arrives, it identifies the flow's behavior, i.e. its function $B(p)$. This flow's behavior is either carried in the packet, or known directly by the tagger when tagging is done at the source.
- The tagger selects a random number X between 0 and 1 and computes the packet's tag $B_i(X)$.

Figure 9 and 10 show how tags are chosen for a TCP flow with a round trip time of RTT=50ms and a CBR flow sending 100 packets per second. Using the $\mathcal{T}$UF filter model with a constant threshold $\hat{K}$, we see on figure 9 that in a stationary state, we get a fair bandwidth sharing, both goodputs being precisely equal to $\hat{K}$. Here is a proof of this theorem, stating that we reach this stable and fair bandwidth sharing.

PROOF :

Since the tags are set according to $Tag_i = B_i(X)$, where X is uniformely distributed between 0 and 1, the tag distribution function F_i is :

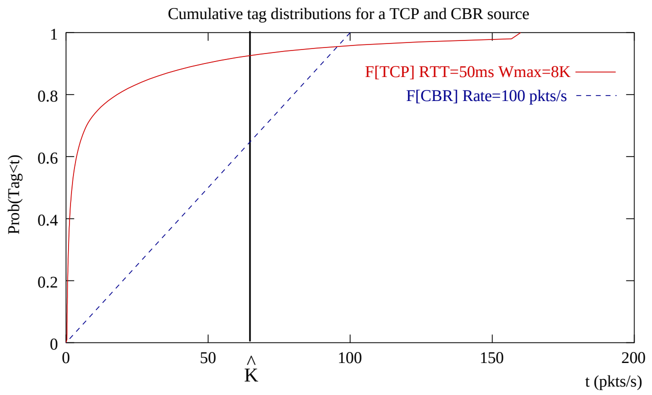

Fig. 10. Tag cumulative distributions for TCP and UDP flows

$$\begin{array}{lcl} [0,\infty] & \xrightarrow{F_i} & [0,1] \\ Y & \mapsto & F_i(Y) = \begin{cases} 1 - B_i^{-1}(Y) & \text{if } Y \le R_i \\ 1 & \text{if } Y \ge R_i \end{cases} \end{array}$$

And thus, $B_i(1 - F_i(Y)) = Min(Y, R_i)$

Following the fluid filter model of the TUF router with a threshold $K(t)$, the downstream rate is the router's capacity C if at least one flow is constrained, $\sum_i R_i$ otherwise:

$$\sum_i T_i.F_i(K) = Min(C, \sum_i R_i) = \hat{C} \qquad (3)$$

The flows' dynamics are modeled by (equations 1, 2):

$$\frac{dT_i}{dt} = \alpha.(\frac{B_i(p_i)}{1-p_i} - T_i) = \alpha.(\frac{Min(K, R_i)}{F_i(K)} - T_i) \qquad (4)$$

Replacing equation 4 in the derivative of equation 3 :

$$\sum_i \frac{dT_i}{dt}.F_i(K) + \sum_i T_i.\frac{dK}{dt}.F_i'(K) = 0$$

$$\frac{dK}{dt} = -\alpha \frac{\sum_i Min(K, R_i) - T_i.F_i(K)}{\sum_i T_i.F_i'(K)}$$

Let $H_i(K) = min(K, R_i)$ and $H(K) = \sum_i min(K, R_i)$. H is a strictly increasing and continous function between 0 and $Max_i(R_i)$. Besides, $H(0) = 0$ and $H(Max_i(R_i)) \geq \hat{C}$. The equation $H(K) = \hat{C}$ therefore has a unique solution $\hat{K}$. Thus,

$$\frac{dK}{dT} = -\alpha \frac{H(K) - \hat{C}}{\sum_i T_i.F_i'(K)} = -\alpha \frac{H(K) - H(\hat{K})}{\sum_i T_i.F_i'(K)}$$

$$\frac{d(K - \hat{K})^2}{dt} = -2.\alpha \frac{(H(K) - H(\hat{K}))(K - \hat{K})}{\sum_i T_i.F_i'(K)} \leq 0$$

$\delta(K) = (K - \hat{K})^2$ is therefore a decreasing - and positive - function. Thus, $(K - \hat{K})^2$ converges and since H strictly increasing, $\frac{d\delta}{dt} \to 0 \Rightarrow K \to \hat{K}$. We can now easily prove that all the flows' sending rate converge:

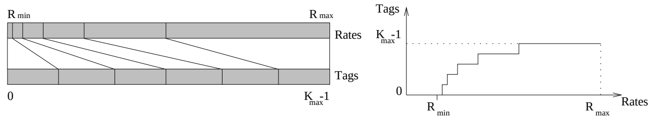

Fig. 11. The tag-rate conversion function

Let $\hat{T}_i = \frac{H_i(\hat{K})}{F_i(\hat{K})}$ and $y = T_i - \hat{T}_i$.

$$\frac{dy}{dt} + \alpha.y = \alpha(H_i(K)/F_i(K) - H_i(\hat{K})/F_i(\hat{K})) \to 0 \qquad (5)$$

One can easily prove that (5) implies that $y \to 0$, i.e. $T_i \to \hat{T}_i$.

To conclude,

- All sending rates converge towards a **stable** value $\hat{T}_i$
- The goodput of flow i is then $\hat{T}_i.(1 - p_i) = \hat{T}_i F_i(\hat{K}) = H_i(\hat{K}) = min(\hat{K}, R_i)$. All constrained flow get the **fair share rate** $\hat{K}$ and unconstrained flow get their maximum throughput R_i.

III. Extensions and implementation issues

A. *The tag field*

We have so far considered tags to be a real value. A packet tagged r will be forwarded through all the routers whose fair share rate (threshold) is at least r. A tag is therefore representative of a rate. To represent a wide range of rates in the Internet while minimizing the uncertainty, we use a logarithmic scale to map rates between R_{min} and R_{max} to discrete integer values between 0 and $K_{max} - 1$. With a 16 bits field, we can map the rates between 2^{-10} and 2^{22} packets per second (i.e. 1 Byte/s to 4 GBytes/s for packets of 1KByte) with an approximation bounded by 0.04%. The general expression of the tag-rate conversion function is (figure 11) :

$$V(r) = \left\lfloor K_{max}.\frac{log(r) - log(R_{min})}{log(R_{max}) - log(R_{min})} \right\rfloor$$

For the specific values chosen above :

$$V(r) = \lfloor 2^{11}.(log_2(r) + 10) \rfloor$$

The tagger actually inserts the integer $V(B(X))$ in the IP packet header. We could add a 16 bits IP option in the IP packet header. This would however significantly increase the packet's processing time. As pointed out in [17], very few packets (0.22%) are actually fragmented. It is possible to use the IP identifier field for the tag when the pair (More Fragment and Fragment Offset) are both set to zero (i.e. the packet is not fragmented). Fragmented packets are ignored by TUF and forwarded as usual. A reserved value of the TOS byte is used to indicate that the packet is transporting a TUF tag.

B. *Tagging TCP and UDP flows*

TCP Flows :

Models of TCP throughput depending on the loss rate and round trip time have been proposed in the literature. The TCP model presented in [18] is a good approximate model for TCP (b is the number of acknowledged packets by a received ACK, and t_0 is the timeout timer) :

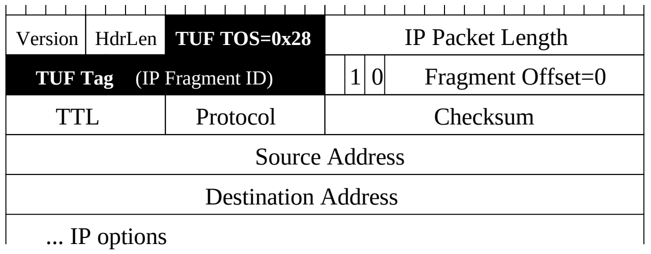

Fig. 12. IP header for TUF packets

$$B(p)/(1-p) = min\,(W_{max}/RTT, \frac{1}{RTT.\sqrt{\frac{2bp}{3}}+t_0.min(1,3\sqrt{\frac{3bp}{8}})p(1+32p^2)})$$

The tagger needs to keep track of the connection's RTT. It updates, exactly as the TCP source, the value of the timeout timer based on the RTT estimations. This becomes very simple if tagging is done at the source, since the source already keeps track of these parameters.

$$Tag_{tcp} = V\,((1-X).min\,(W_{max}/RTT, \frac{1}{RTT.\sqrt{\frac{2bX}{3}}+t_0.min(1,3\sqrt{\frac{3bX}{8}})X(1+32X^2)}))$$

UDP Flows :

Estimating the goodput as a function of the loss rate is very simple for a CBR source whose rate R is known:

$$\begin{aligned} B_{cbr}(p) &= R.(1-p) \\ Tag_{cbr} &= V(R.(1-X)) \end{aligned}$$

This formula can be used for unresponsive flows, considered equivalent to CBR sources over short time periods. The tagger must estimate the rate R of the flow. For that purpose, it updates the rate estimation using an exponential averaging. The rationale for using this kind of averaging has been discussed in [12].

$$R_{new} = (1-e^{-\Delta t/K})/\Delta t + e^{-\Delta t/K}.R_{old}$$

where Δt is the packet interarrival time, and $K = 100ms$.

Other Flows :

For responsive flows, that use their own congestion control protocol, we propose three methods to set the tag :

- Tag is set at the source.
- The behavior of the end-to-end congestion protocol is described in an IP option, used and then removed by the tagger. Packets follow the slow path only between the source and the tagging edge router.
- A new IP protocol can be defined, with its own IP protocol number. Information regarding the end-to-end behavior can then be carried in the first bytes of this protocol's header, and is used for the tagging.

C. Improving the loss pattern

The tagging function presented in section II-E uses a stochastic variable X uniformly distributed between 0 and 1. This can be a random value. However, by choosing the value of X (still uniformly distributed between 0 and 1) so as to interleave low tag values with high tag values, we can improve the loss pattern and avoid bursts of losses. Since B is a decreasing function, low tag values are obtained with values of X close to 1, and high tag values with values of X close to 0. We propose to choose the sequence of values X_n taken by X for a flow as follows :

The value of the variable X for packet n from flow F is $X_n = I(n + D_F)$, where D_F is a random integer, used to avoid flow synchronization, and :

$$I : \begin{array}{ccc} \mathbb{N} & \longrightarrow & [0,1[\\ \sum_{i=0}^{\infty} b_i.2^i & \mapsto & \sum_{i=0}^{\infty} b_i.2^{-i-1} \end{array}$$

For example, with $D_F = 1$, X would take the following values : $1/2, 1/4, 3/4, 1/8, 5/8, 3/8, 7/8, 1/16, \ldots$, alternating high and low tags.

D. Incremental deployment

TUF can be incrementally deployed, since :

- Non-tagged packets can traverse TUF routers. Minimum bandwidth should be reserved for these packets. We can use two separate queues served in a weighted round-robin manner : the queue with tagged packets, and the queue with non-tagged packets.
- Tagged-packets can go though non-TUF routers. Of course, fair bandwidth sharing is not assured in non-TUF routers.

E. Multiple congested routers

Our mechanism is clearly not affected by the presence of non-congested routers along the flow's path, since these do not change the tags and do not drop packets. But it is also interesting to notice that having multiple congested routers along a flow's path is equivalent to having only the bottleneck router congested. Indeed serializing low-tag filters is equivalent to having only the filter with the lowest threshold. However, as the number of hops increases, modeling the succession of TUF routers as one TUF filter becomes more and more approximate. Fortunately, simulations show our algorithm remains satisfactory, even when the number of hops reaches 20.

F. TUF and layered encoding

If tagging is done at the source, layered encoding applications can benefit from TUF by having packets from the higher layers dropped first. Suppose that our application sends its data into three layers, at the same rate in each layer. To send data in the first layer, the application chooses X uniformely distributed in the interval $[2/3, 1[$ and computes the tag. For the second and third layers, X is uniformely distributed in the intervals $[1/3, 2/3[$, and $[0, 1/3[$. As long as the application sends at an equal rate in each layer, the overall X is uniformely distributed between 0 and 1, and we achieve fairness. The packets dropped first will be those of the last layer (corresponding to low values of X). This can be generalized to any number of layers with various rates.

IV. Simulations

In this section, we evaluate our proposal with simulations done using the $ns-2$ simulator [19]. In these simulations, we used the following algorithms to compare the throughputs and fairness achieved :

- TUF : Our Tag-based Unified Fairness scheme.
- TUF_{udp} : A diminished version of TUF , where all flows are tagged as unresponsive UDP flows.
- CSFQ : Core Stateless Fair Queueing [12]. We used the released code of CSFQ for the ns simulator [20].
- FIFO : The standard FIFO router
- RED : The Random Early Discard router [15].
- DRR : The Deficit Round Robin router [11]. DRR serves as a reference in terms of fairness for our simulations.
- SFQ : The Stochastic Fair Queueing router. Another implementation of Fair Queueing.

TUF_{udp} is the **s**ame algorithm as TUF , except all flows are tagged as unresponsive flows. We introduce this scheme to show that not taking into account the responsive nature of the flow is a cause of inefficiency in existing stateless schemes, as explained in section II-B. Indeed, as we show below, TUF , although stateless, has similar performances than DRR, SFQ, and better performances than other stateless schemes TUF_{udp} , CSFQ[12], and thus RFQ [13], ...

In the two first set of simulations, we reproduced some of the scenarios presented in [12] for better comparison. In these set of simulatons, we prove the correct Fair Queueing behavior of TUF in non-hostile homogeneous environments, where round trip time (RTT) is low (2ms), and traffic is not bursty. In the third set of simulation, we introduce larger round trip times and bursty traffic that have a serious impact on responsive flows. TUF is the only stateless fair queueing algorithm that maintains fairness in these environments.

By default in our simulations, links have a capacity of 10 Mbps, and a propagation delay of 1ms. Our buffers length is 64 KBytes, and all packets are 1000 bytes long (ns default values). For DRR and SFQ, we set the number of buckets to 1024. The queue size for SFQ is set to 2MBytes. The default simulation time is 60 seconds.

A. A single congested link

The first set of simulations is run with the topology shown on figure 13. We evaluate the fairness of the different mechanisms when a number of UDP and TCP connections share the same link.

In the first simulation (figure 14), 32 UDP flows with varying aggressivity share the same 10Mbps bottleneck link. The arrival rate for the UDP flow number i is $(i+1)$ times the fair share rate, i.e. $(i+1) * 10Mbps/32$. Figure 14 shows the normalized bandwidth[1] achieved by all the UDP flows. The RED and FIFO queueing disciplines do not ensure fairness, and the most aggressive flows (with the greatest flow IDs) achieve the best throughputs. On the other hand, SFQ, DRR, CSFQ, and TUF propose comparable output rates to all the UDP flows.

The second simulation (figure 15) evaluates the impact of an ill-behaved UDP flow (Flow ID=0) on a set of 31 TCP connections. The UDP flow is a CBR source whose rate is the link's capacity, i.e. 10Mbps. Again, apart from RED and FIFO, that give most of the bandwidth to that flow, SFQ, DRR, CSFQ and TUF limit the rate of the UDP connection to its fair share, as all the other connections.

[1] the ratio between the goodput and the fair share rate

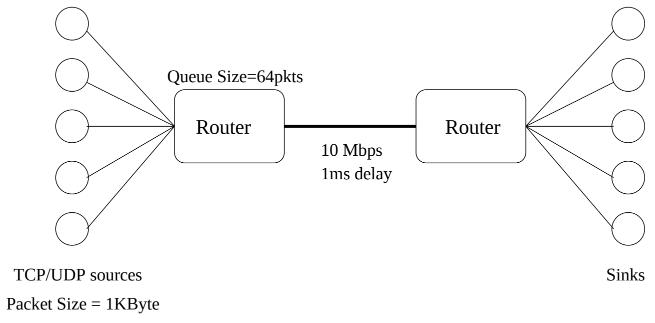

Fig. 13. The single congested link topology

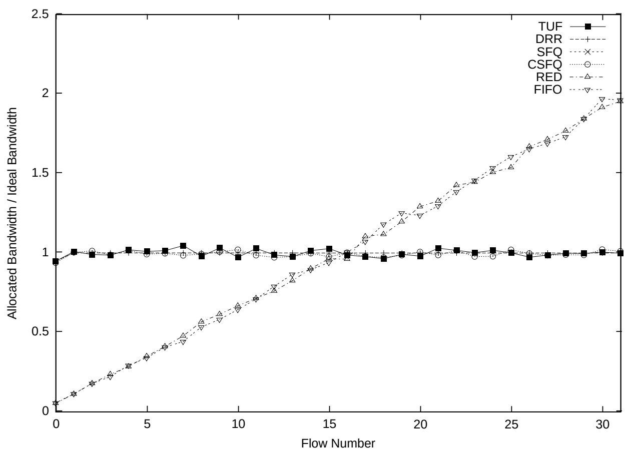

Fig. 14. 32 CBR flows sharing a 10 Mbps links.

In the third simulation (figure 16), we evaluate the normalized bandwidth of a single TCP connection subject to the pressure of an increasing number of concurrent UDP connections. The UDP connections all send at twice the fair share rate. Note that, as explained in [12], the performances of DRR are significantly affected when the number of flows exceeds 22, because of the limited buffer space reserved for the TCP connection. Although all fair queueing algorithms propose reasonable performances for the TCP connection (that receives at least 60% of its fair share rate), TUF_{udp} and CSFQ propose a lower bandwidth than TUF , DRR, or SFQ. This is typically a symptom of the *variation effect*, which, for TCP-Reno, would limit the normalized bandwidth to 75%.

B. Multiple congested links

The second set of simulations is run with the topology shown on figure 17. The purpose of these simulations is to evaluate the robustness of the algorithms when flows traverse more than one congested link. 10 cross CBR sources send at 2 Mbps on each of the congested links. This cross traffic enters the path in one of the router and exits at the next. The main source is a TCP source or a UDP source sending at its fair share rate (909Kbps).

The first simulation (figure 18) with a TCP connection

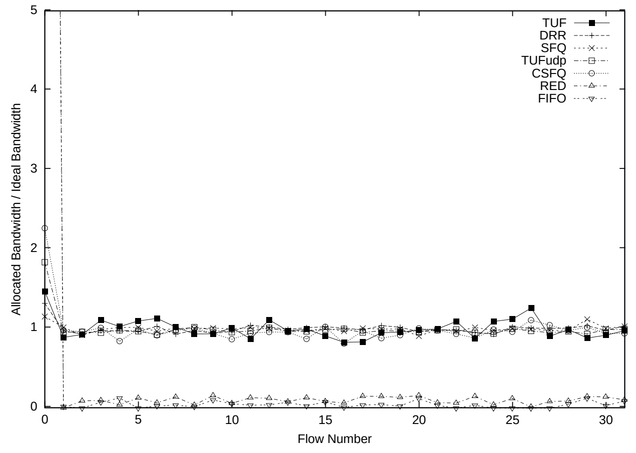

Fig. 15. A 10 Mbps CBR flow (ID=0) sharing the link with 31 other TCP flows.

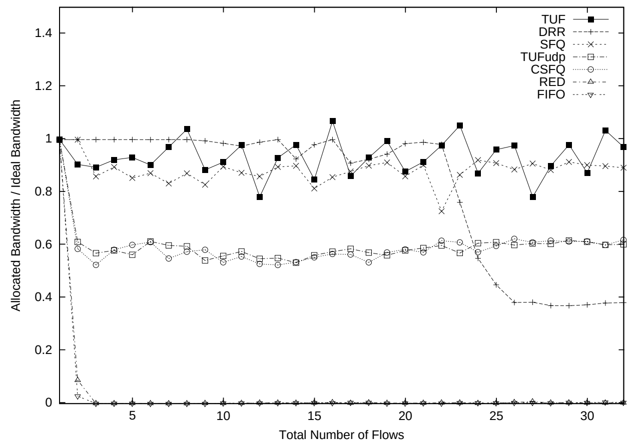

Fig. 16. TCP flow sharing the link with 0..31 UDP flows

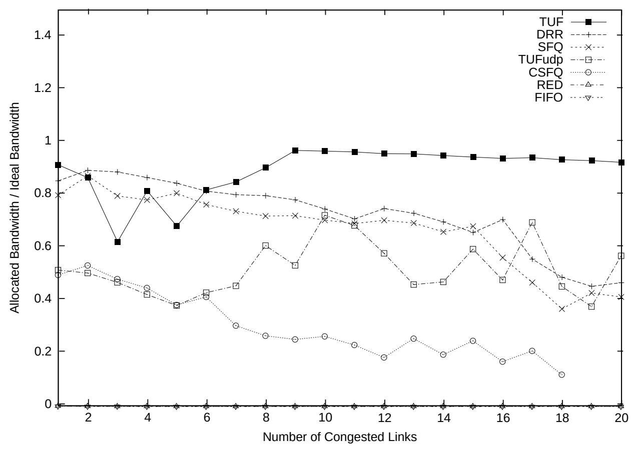

Fig. 18. TCP connection through N=1..20 congested links

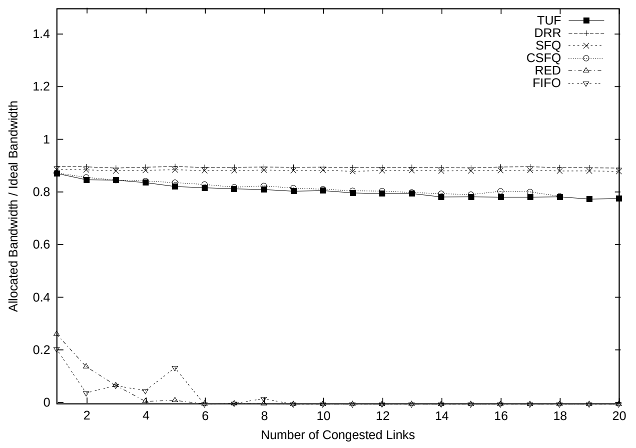

Fig. 19. CBR connection through N=1..20 congested links

demonstrates the robustness of TUF to multiple congested links. As in figure 16, TUF_{udp} and CSFQ propose reasonnable fairness (around 60%) that rapidly decreases in the case of CSFQ. Surprisingly, DRR and SFQ performances also degrade as the number of hops increases. RED and FIFO routers do not enable the TCP connection to achieve a significant throughput.

In the second simulation (figure 19), the main UDP source is not affected by the cross traffic, and achieves perfect fairness in all scenarios (TUF , CSFQ, SFQ and DRR) , whereas RED and FIFO are unable to maintain a significant throughput. (Note that TUF and TUF_{udp} are equivalent here). SFQ and DRR very slightly improve the fairness compared to TUF and CSFQ. This

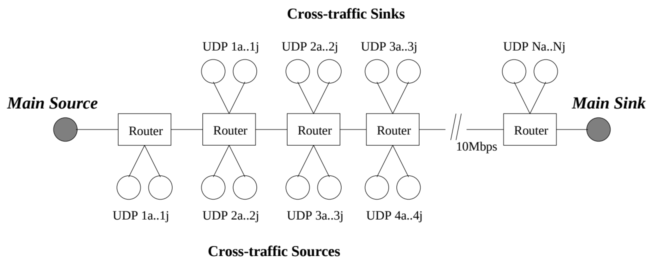

Fig. 17. The multiple congested link topology

is probably due to the slight inaccuracy of the flow rate estimations, that do not exit in SFQ and DRR.

C. *Heterogenous environments*

This third set of simulation shows the vulnerability of network mechanisms that try to adjust instantaneous rates, such as stateless fair queueing algorithms, in heterogeneous environment. Hereby, we want to show that taking into account the end-to-end congestion control protocol is mandatory to obtain fairness. We here put emphasis on the *burst effect* presented in section II-B : in the first simulation through the introduction of large round trip times, in the second with bursty cross traffic that cause large variations in the fair share rate.

C.1 Large round trip times

In this simulation (figure 20), we use the single link topology (figure 13). 8 CBR flows, sending at twice their fair share rate, share the link with a TCP flow, whose round trip time varies from 2ms, as in the previous simulations, to 1s. Figure presents the normalized bandwidth achieved by the TCP flow. Due to the very large round trip time and to have significant statistics on the losses, the duration of the simulation was significantly increased to 10000 seconds.

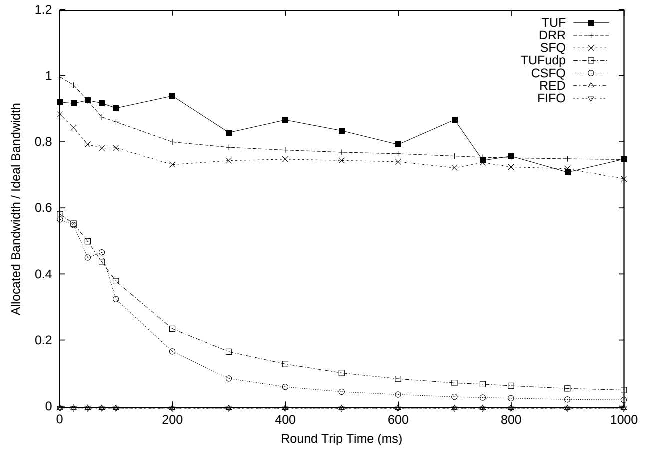

Fig. 20. TCP flow round trip times and fairness

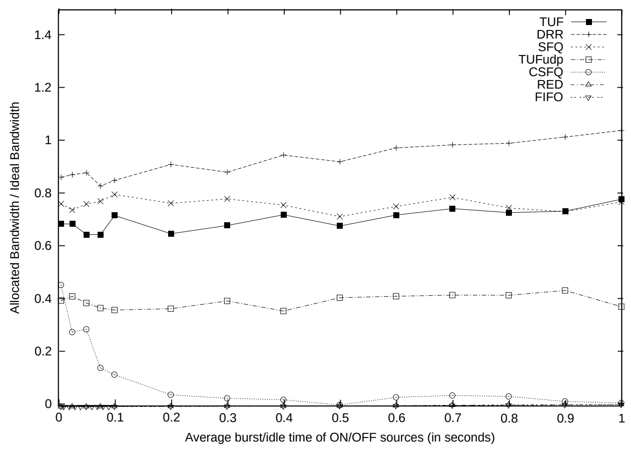

Fig. 22. TCP flow subject to bursty cross traffic.

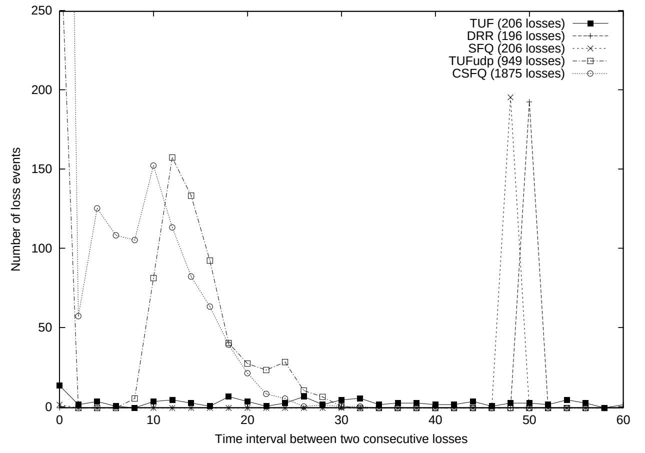

Fig. 21. Distribution of the interval between consecutive losses

Figure 21 presents the distribution of the time interval between two consecutive losses. The X-axis represents the time interval and the Y-axis the number of loss events. These graphs confirm that losses are very bursty for CSFQ and TUF_{udp} : the number of losses are 5 to 10 times more important, and an important proportion of losses occur just after another loss. This is not the case for TUF , DRR or SFQ.

In figure 20 we see the impact of this *burst effect* on fairness. It is particularly interesting to notice the difference between the fairness of TUF_{udp} and TUF , since these implement the same algorithm except that the only the latter takes into account the nature of the TCP flow. TUF_{udp} performances, as well as CSFQ performances, become significantly low when the round trip time increases. DRR and SFQ that maintain average rates using flow state realize performances similar to those of TUF .

C.2 Bursty cross traffic

This simulation is run on the multiple congestion link topology (figure 17) with 5 congested link. The CBR sources that formed the cross traffic are now replaced with ON/OFF sources. The burst (ON) and idle (OFF) time periods are both exponentially distributed with the same average chosen between 5ms and 1s. The cross traffic's average intensity is the same than for the previous set of simulations : the UDP ON/OFF sources send at 4 Mbps during the ON period.

Figure 22 once again depicts the poor performances of TUF_{udp} or CSFQ compared to that of TUF , DRR and SFQ. In CSFQ, performances become particularly poor when we reach 100ms for the burst and idle time, which precisely corresponds to the averaging window for the "instantaneous" rate estimation.

V. Implementation

We implemented TUF in Linux version 2.2. Tagging is done in the source, at the IP layer, for all packets whose TOS byte is 0x28, the TOS value chosen for TUF packets. It is therefore transparent for the application. We consider only two flow behaviors : TCP flows and unresponsive UDP flows. For UDP packets, state is maintained at the source on a per socket basis to evaluate the flow's rate. For TCP packets, the round trip time, the retransmission timer, and the maximum window size are already evaluated by the kernel. The computation of a TCP or UDP tag requires less than 90 elementary operations (addition or multiplication). We experimented our TUF algorithm on a small Y-topology. We introduced greedy CBR flows, TCP flows, and non greedy flows, such as RAT[2] audio flows. As we activate the queueing discipline in the core router, the throughput achieved by the TCP flow and the quality of the audio session become significantly better, and fairness is obtained with the other CBR flows. Experimenting TUF on a larger scale would be interesting.

VI. Conclusion

In this paper, we presented TUF , our Tag-Based Unified Fairness algorithm. Our scheme enables us to do fair bandwidth sharing among flows of different types, such as TCP and greedy UDP flows, without requiring flow-state in the high-speed backbone routers. Packets are tagged at the edge of the network, or at the source, with a value that represent the minimum fair share rate a router must support to forward the packet. The queueing discipline in the TUF router is very simple and consists in dropping the highest tag value when the queue is full. This behaves as a bufferless filter, that lets through packets with tag

[2] http://www-mice.cs.ucl.ac.uk/multimedia/software/rat

below a threshold K. We simulated TUF , and it achieves approximately fair bandwidth sharing, as CSFQ, DRR and SFQ. However, it adapts specifically to any form of responsive flow whose throughput can be determined as a function of the loss rate. In heterogenous environments, with non-negligible round trip times or bursty traffic, it thus provides much better fairness than other stateless fair queueing algorithms that try to adapt instantaneous rates, such as CSFQ or RFQ. We believe that, by its simplicity and efficiency, TUF is an interesting approach to a hybrid network and end-to-end congestion control.

REFERENCES

[1] D.M Chiu and R.Jain, “Analysis of the increase and decrease algorithms for congestion avoidance in computer networks,” *Computer Networks and ISDN Systems*, vol. 17, pp. 1–14, 1989.

[2] Sally Floyd and Kevin Fall, “Promoting the use of end-to-end congestion control in the internet,” *IEEE/ACM Transactions on Networking*, August 1999.

[3] W. Tan and A. Zakhor, “Real-time internet video using error resilent scalable compression and tcp-friendly transport protocol,” *IEEE Transactions on Multimedia*, vol. 1, no. 2, pp. 172–186, June 1999.

[4] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer, “Equation-based congestion control for unicast applications,” in *Proceedings of ACM SIGCOMM’2000*, May 2000.

[5] S. Jamaloddin Golestani and Krishan K. Sabnani, “Fundamental observations on multicast congestion control in the internet,” in *Proceedings of IEEE Infocom’99*, March 1999, pp. 990–1000.

[6] Mark Handley and Sally Floyd, “Strawman specification for tcp friendly (reliable) multicast congestion control (tfmcc),” proposed to the RM mailing list, November 1998.

[7] Steven McCanne, Van Jacobson, and Martin Vetterli, “Receiver-driven layered multicast,” in *Proceedings of ACM SIGCOMM’96*, 1996, pp. 117–130.

[8] Alan Demers, Srinivasan Keshav, and Scott Shenker, “Analysis and simulation of a fair queueing algorithm,” in *Proceeding of ACM SIGCOMM’89*, 1989, pp. 3–12.

[9] Jon C.R. Bennett and Hui Zhang, “Wf2q : Worst-case fair weighted fair queueing,” in *Proceedings of IEEE Infocom’96*, San Francisco, CA, March 1996, pp. 120–128.

[10] Jon C. R. Bennett and Hui Zhang, “Hierarchical packet fair queueing algorithms,” in *Proceedings of ACM SIGCOMM’96*, October 1996, vol. 26, pp. 143–156.

[11] George Varghese, “Efficient fair queuing using deficit round robin,” in *Proceedings of ACM SIGCOMM’95*, 1995, vol. 25, pp. 231–243.

[12] Ion Stoica, Scott Shenker, and Hui Zhang, “Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks,” in *Proceeding of ACM SIGCOMM’98*, 1998, vol. 28, pp. 118–130.

[13] Zhiruo Cao, Zheng Wang, and Ellen Zegura, “Rainbow fair queueing: Fair bandwidth sharing without per-flow state,” in *Proceedings of IEEE Infocom’2000*, March 2000.

[14] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis, “Choke, a stateless active queue management scheme for approximating fair bandwidth allocation,” in *Proceedings of IEEE Infocom’2000*, March 2000.

[15] Sally Floyd and Van Jacobson, “Random early detection gateways for congestion avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[16] Dimitri Bertsekas and Robert Gallager, *Data Networks*, chapter 6, pp. 524–529, Prentice-Hall, 1987.

[17] Ion Stoica and Hui Zhang, “Providing guaranteed services without per flow management,” in *Proceedings of ACM SIGCOMM’99*, 1999.

[18] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose, “Modeling tcp throughput: A simple model and its empirical validation,” in *Proceedings of ACM SIGCOMM’98*, 1998.

[19] Steve McCanne and Sally Floyd, “Ucb/lbnl/vint network simulator (ns) 2.1b5,” http://www-mash.cs.berkeley.edu/ns/.

[20] Ion Stoica, “Csfq simulation scripts for ns-2,” http://www.cs.cmu.edu/ istoica/csfq/, 1998.