



# A lightweight marker with partial state information for DiffServ networks <sup>☆</sup>

Claudio Casetti <sup>\*</sup>, Marco Mellia

*Dipartimento di Elettronica, Politecnico di Torino, Torino, Italy*

Received 29 July 2004; received in revised form 29 July 2004; accepted 6 December 2004  
Available online 7 March 2005

Responsible Editor: J. Sole-Pareta

---

## Abstract

A problem common to packet markers used in DiffServ architectures is the provision of Quality of Service to TCP flows: specifically, short-lived TCP flows suffer from packet losses at small congestion windows, which inevitably leads to lengthy retransmission timeouts. It would therefore be sensible to mark packets of short-lived flows so as to offer them a greater level of protection. In this work, we propose a class of Fair Markers that, without the penalty of per-flow management, achieves the same performance enhancement of a per-flow marker, but with a much simpler design and limited implementation complexity. Extensive simulations with realistic traffic scenarios and simple analytical models allow us to compare our proposal with existing ones. We also underline the importance of providing a minimum protection to both data and acknowledgment segments of traffic crossing congested domains.

© 2005 Elsevier B.V. All rights reserved.

*Keywords:* DiffServ; TCP; Packet marking; Fairness

---

## 1. Introduction

The current Internet provides a best-effort service, in which routers treat all packets in a uniform fashion. There have been several efforts to enhance

the Internet and support Quality of Service (QoS) and service differentiation: namely, IntServ [1], and DiffServ [2]. Scalability issues have questioned the effectiveness of IntServ, which requires per-flow signaling and per-flow guarantees. On the contrary, DiffServ handles flow aggregates, performing packet classification into coarse classes at the network ingress and supporting different per-class guarantees at every hop in the network core. Within the DiffServ framework, the user and the network

---

<sup>☆</sup> This work was supported by the European network of excellence Euro-NGI.

<sup>\*</sup> Corresponding author.

*E-mail address:* [casetti@mail.tlc.polito.it](mailto:casetti@mail.tlc.polito.it) (C. Casetti).

establish a *Service Level Agreement* (SLA), i.e., a contract that states the QoS requirements of the user and the allocation of resources the network sets aside to that user. The SLAs are based on aggregate, simple descriptions of the traffic, and are not meant to be used on a per-flow basis.

DiffServ comes in three flavors: classic *Best Effort*, for which no particular mechanism is adopted to guarantee any level of QoS; *Expedited Forwarding* (EF) [3], which is also called ‘Virtual Wire’, providing an almost airtight separation between premium traffic (i.e., traffic using the EF class of service) and non-premium traffic; *Assured Forwarding* (AF) [4], in which different classes are given different forwarding and dropping treatments, although they share the same network resources. The main idea behind the AF class of service (or Per Hop Behavior) is to differentiate packets marking them according to a given throughput profile. When the flow aggregate conforms to a *Committed Information Rate* (CIR) (with the possible addition of a temporary, limited surplus of bandwidth), then packets will be marked as *in-profile*. Otherwise, when the flow aggregate is non-conformant to the CIR, packets will be marked as *out-profile*. Once packets have been classified at edge network nodes, they will be treated by core nodes using differentiated dropping probabilities, so the in-profile packets will be discarded only after all out-profile traffic has already been dropped. At network edge, packet markers such as the *Single- or Two-Rate Three-Color Marker* [5,6] (TCM) or the *Time Sliding Window Three-Color Marker* [7] (TSWTCM) must be used to mark packets. Traditionally, packet marking follows a “color code,” so that within the same SLA “green” packets receive a better service than “yellow” packets, which receive a better service than “red” packets, managed as Best Effort traffic. In this paper we will refer to in-profile traffic as *Green*, while *Yellow* class will be used to represent both out-profile and Best Effort traffic. Both markers consider the packet flows belonging to an SLA as pure aggregate of packets, and do not distinguish them based on higher-layer protocols.

The AF class offer the sort of soft QoS guarantees that is best suited to the characteristics of data transported using TCP. Thus, it is not surprising

that many studies have shown the benefits and drawbacks of the DiffServ scheme for long-lived TCP traffic [8–12], and for HTTP-like traffic [13–16], and that different markers have been proposed.

A common problem of packet markers, underlined in previous works, is their inability to provide a similar level of protection to long- and short-lived flows alike. In particular, short-lived flows, representing the majority of today’s traffic, suffer from packet losses occurring during or just past the three-way handshake phase, when the TCP congestion window size may not be large enough so as to trigger the Fast Recovery algorithm. Such flows are delayed by (possibly) repeated retransmission timeouts that are affecting them only because of their state, while longer flows manage to avoid timeouts thanks to the reception of duplicate ACKs that enable the Fast Recovery mechanism. A similar situation occurs right after a timeout, when a flow is more vulnerable because of its small window, and the danger of repeated timeouts caused by erroneous packet marking which may lead to multiple drops is consistent. This gives rise to unfairness issues, where long-lived flows manage to obtain the largest benefits from the adoption of a DiffServ scenario.

Given the importance of the TCP protocol and of Web traffic, many proposals addressing this problem have been presented in the literature [10,12,14,15,17]. In particular, they show that it would be desirable to design *Fair Markers* so that they are (at least partially) aware of the state of TCP connections. Such awareness could translate into differential marking of flows that have been running for the longest time, improving a short-lived flow chance of using a larger amount of green packets, therefore leaving the small-window regime without suffering from packet drop. However, all previous work requires either to modify the TCP stack at each host [10], or to maintain a per-flow state at network edges [12,14,15]. The main contribution of this paper is the proposal of a Fair Marker that, by using a limited amount of CPU and memory in edge nodes, achieves the same performance enhancement of a per-flow marker, but with a much simpler design.

In addition, all previous results consider TCP flows that cross forward-congested links, while

assuming that the reverse path is generally not congested. While this can be true considering a single domain, this is not generally the case when multiple domains are crossed. A further contribution of our paper is the investigation of the impact of congestion on the reverse path, showing that a bidirectional SLA is necessary to best exploit the QoS enhancements made possible by the adoption of the DiffServ technology.

In the remaining of the paper we will investigate, through the extensive use of simulation, the QoS received by single, short-lived flows (Section 2), then address the problem of traffic crossing a backward congested path and issues of per-flow marking in Section 3. In Section 4 we describe the Fair Marker architecture; finally we provide simulation results of its performance as well as configuration issues in Section 5, while in Section 6 analytical models will show the benefits of the packet marking architecture. Section 8 summarizes our findings.

## 2. Simulation scenario

To correctly identify the set of problems we wish to solve, we will present simulation results early on in the paper. Results were obtained using *WETMO* (WEB Traffic MOdule) [18], a novel traffic generator model that has been implemented in *ns* and is designed to simulate realistic Web traffic with flows of different lengths exchanged among clients and servers, mimicking the dynamics of HTTP requests and replies. The complexity of network topologies that can be simulated by *WETMO* is limited only by the availability of memory resources. *WETMO* can be used to investigate the performance of DS-enabled Web-like traffic that crosses both forward and backward congested domains. It is thus necessary to first outline the simulation settings and the traffic model that were used throughout this work.

### 2.1. Network scenario

Simulations aim at evaluating the performance of pairs of client–server clouds over a DiffServ domain. Specifically, two different topologies were

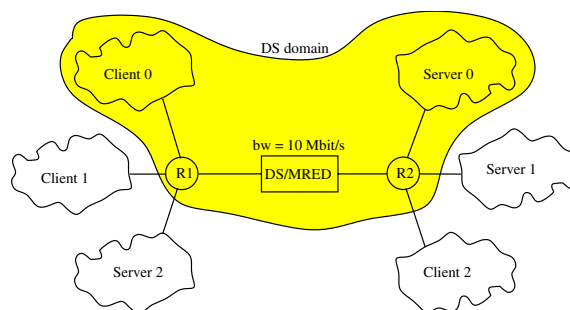


Fig. 1. Single-bottleneck network scenario.

investigated. The first, showed in Fig. 1, features a single bottleneck connecting client–server clouds. The pairings identify client–server relationships, their connection data being carried over a 10-Mbit/s bottleneck link, characterized by a 20 ms delay. At each end of the bottleneck link, two edge routers mimic ingress/egress nodes (depending on the direction of traffic flows) of a DiffServ domain: they are marked R1 and R2 in the figure. Each router receives traffic from the attached clouds in the form of aggregate TCP connections and marks these segments according to the specific SLA; then, following the DS guidelines, it places packets in one virtual queue. Router queues use Multiple-RED (MRED) active management techniques, in which several virtual RED queue management algorithms impose different dropping probabilities to packets according to their marking.

In the scenario in Fig. 1 (referred to as the “single bottleneck” scenario), two client clouds (Client 0 and Client 1) are attached to R1 and one client cloud (Client 2) is attached to R2; server clouds are correspondingly attached to the opposite edge node. Two client clouds connected to R1 send requests to servers connected to R2. Client 1/Server 1 traffic will always send Best Effort (BE) traffic, while the Client 0/Server 0 traffic will be configured so that (i) servers and clients use the BE class, (ii) servers send DS traffic, while clients only send BE packets, or (iii) both servers and clients use a portion of DS-reserved bandwidth.

A third BE cloud pair (with Server 2 connected to R1 and Client 2 to R2) is added to simulate possible congestion on the reverse path: server data of the third cloud pair are routed over the same path

as client requests from the other two pairs. Each cloud includes one router with multiplexing/marketing functionalities, to which all active sources are connected, managing traffic either coming from servers or coming from clients. These routers are linked to the edge routers by a 1-Gbit/s link, with a 1- $\mu$ s latency, so they are never congested.

The second, more complex topology (referred to as the “multi-bottleneck” scenario) is depicted in Fig. 2. Here, the competition between Web traffic and other types of traffic is stressed by the introduction of CBR sources using UDP transport mimicking multimedia streams. Also, the topology features the presence of cross-traffic, of both DS and BE nature, at intermediate routers. The client-server associations, as well as their topology, are indicated by numbers and by the ‘DS’ and ‘BE’ tags. Each bottleneck link has a 10-Mbit/s bandwidth, while access links have a 1-Gbit/s bandwidth, like in the single-bottleneck scenario.

2.2. Queueing disciplines

To perform the active management of the queues at edge routers, we have chosen *MRED* (*Multi-level Random Early Detection*). Only one physical queue is implemented in each ingress of the router, and this queue is split into two virtual

queues: in order of priority, one for segments marked as *Green*, and one for *Yellow* segments and Best Effort (unmarked) ones. With this choice, only one average queue dimension is used.

Each physical queue is managed according to the *FIFO* (*First In First Out*) discipline, therefore the sharing within a virtual queue is only meaningful to MRED.

All simulations were performed using the MRED staggered parameter set depicted in Fig. 3. The reason behind the choice of a staggered parameter set rather than an overlapped one lies in its property of lower drop preference protection:

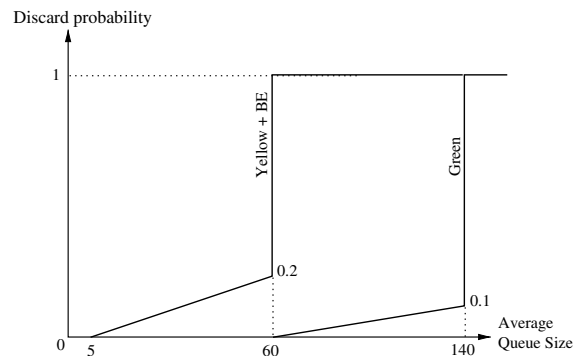


Fig. 3. MRED parameters for edge routers.

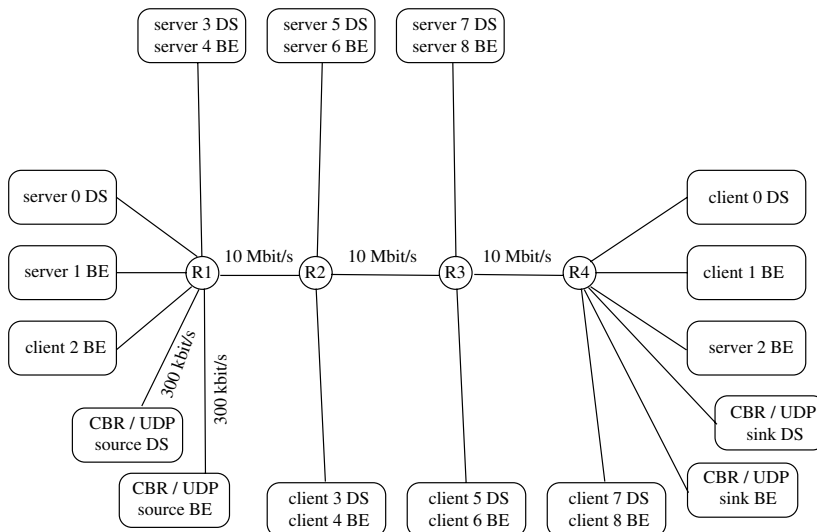


Fig. 2. Multi-bottleneck network scenario.

*Green* segment drop begins after all *Yellow* segments have been dropped. Simulations were done configuring MRED with the following boundary staggered set:

- maximum queue size equal to 200 segments;
- $\min_{th}$ ,  $\max_{th}$  and  $\max_p$  for *Green* traffic equal to 60 segments, 140 segments and 0.1;
- $\min_{th}$ ,  $\max_{th}$  and  $\max_p$  for *Yellow* and BE traffic equal to 5 segments, 60 segments and 0.2.

The parameters we chose reflect those suggested in the literature [19]. We chose to not explore the RED/MRED parameters set because of its size. Indeed, considering the impact of RED/MRED parameters will require too much space, and it is out of the scope of this paper.

### 2.3. Traffic model description

The traffic model was designed to mimic HTTP flows as realistically as possible, comprising requests from clients and replies from servers, each connection being simulated since the three-way handshake until its termination, using the New-Reno Full-TCP *ns* agent. Flow scheduling times and request/response sizes are chosen as close as possible to an actual client/server Web-like transaction.

In order to realistically simulate the size distribution of HTTP flows, we define two distributions of possible flow lengths, one for client requests and another for server replies. In each distribution, the flow lengths are indicated as number of packets to send (P2S), which can be computed from the size in bytes when the segment size is set. For the sake of simplicity, each segment across the network is full-sized at 1000 bytes: therefore the P2S value, obtained dividing the size in bytes by the segment size, must be approximated in excess.

Following the methodology already used in [13,14], client requests consist of a number of segments which can vary from one to three according to the distribution shown in Table 1, where the percentages indicate the incidence on the total of the corresponding flows. Many requests have only one segment because today the greatest number of HTTP interactions between client and server con-

Table 1  
Lengths of client requests

	P2S	Share of the total (%)
	1 PKT	85
	2 PKT	10
	3 PKT	5
Average	1.2	100

Table 2  
Lengths of server responses

	Flow lengths in bytes	P2S
	61	1 PKT
	239	1 PKT
	539	1 PKT
	1349	2 PKT
	2739	3 PKT
	4149	5 PKT
	6358	7 PKT
	10910	11 PKT
	19878	20 PKT
	90439	91 PKT
Average	13666	14.2 PKT

sist of HTML page requests, that require only a small number of bytes. The flows with two or three segments simulate particular requests needing more bytes, for example, requests of active pages where the user submits client-side information.

Once a request has been completely received from a server, the size of the corresponding reply is chosen in a randomly uniform fashion among the flow lengths listed in Table 2, that was derived in [13,14] to best match real traffic distribution.

At the client side, each DiffServ cloud generates Web request flows that follow a Poisson process: requests are separated by an exponentially-distributed random delay, whose average value is computed considering:

- $B$  as the bandwidth of the bottleneck link, expressed in Mbit/s;
- $\rho$  as the ratio of the total offered load (including TCP and IP headers) to the bottleneck bandwidth;
- $\alpha_i$  as the fraction of global traffic generated by cloud  $i$ ,  $\sum_i \alpha_i = 1$ .

It is straightforward to observe that, with the flow lengths listed above, the more heavily loaded direction of the path is the one from server to client (with an average of 13.7 kB per flow, as opposed to only 1.2 kB per flow in the opposite direction).

In the single-bottleneck scenario we fixed the total server load ( $\rho$ ) at 80% of the bottleneck bandwidth, i.e., at 8 Mbit/s, and we varied the DS load ( $\alpha_{DS}$ ) between 60% and 95% of the overall traffic. The CIR on the server-to-client path is set to 6.4 Mbit/s, i.e., 80% of the total offered load. Similarly, in the multi-bottleneck scenario, we fixed the server loads so that, on each bottleneck link (i.e., between routers  $R_i$  and  $R_{i+1}$ ), on either direction, the load offered by the server amounts to 80% of the available bottleneck bandwidth. Considering that UDP traffic offers an average load of 600 kbit/s, on the left-to-right path the total load is 7.52 Mbit/s (80% of 9.4 Mbit/s not used up by UDP traffic), while it is 8 Mbit/s on the right-to-left path, since there is no UDP traffic.

This bandwidth allocation allows us to identify two operating regions, one where the DS load is below the CIR (*SLA overprovisioning*), and one where the DS load is above the CIR (*SLA underprovisioning*). In the rest of the paper, all plots report a vertical dotted line separating the two regions.

Considering the client-to-server path for DS flows, we tested two cases: one where no explicit reservation is performed, and one where client flows are allocated a 560 kbit/s CIR, which in our traffic model corresponds to about 80% of the client offered load.

In both scenarios, the third pair (pair 2), i.e., the one that generates traffic to load the edge routers on the reverse path, always activates BE connections so that the traffic it generates reaches an average rate of 8 Mbit/s.

We also ran simulations where we vary the total offered load  $\rho$ , maintaining the DS load at a fixed rate. These results are not included in this work, but they can be found in [20]. They largely confirm the same findings we include in this paper.

Finally, simulation of bursty, short-lived flows need long runs in order to be confident with the obtained results. In our study, to get accurate results, each simulation is run for at least 600 s, so

that the initial transient phase has little impact on the results, and results are then averaged over 8 independent runs.

### 3. TSW2CM evaluation and drawbacks

Previous research has pointed out that the Diff-Serv architecture cannot fully guarantee QoS requirements to TCP flows, and, in particular, to Web-like file transfers.

In this section we briefly summarize the benefits and drawbacks of having an SLA that protects a DS source from competing BE traffic, using classic markers. We select as main QoS performance index the *Completion Time* (CT) of an HTTP transaction, that takes into account the amount of time required to successfully receive a server reply. It is built by three components: (i) TCP three-way-handshake during connection setup, (ii) client requests to the server, (iii) server replies to the client. We do not consider the closing procedure (even if it is simulated) since it does not affect the CT perceived by the user.

In order to evaluate the classic DS behavior, the marking strategy we selected is the standard TSWTCM [7], used in two-color mode (i.e., by setting  $PIR=CIR$ ): therefore, packets are only marked as either *Green* or *Yellow*, and not *Red*; in this paper we will refer to it as TSW2CM.

The TSW2CM consists of two independent components: a “rate estimator,” and a random “marker” to associate a color, amounting to a drop precedence, to each packet. Fig. 4 reports a simple illustration of the TSW2CM, in which the two blocks are identified.

Every time a new packet transits, the rate estimator executes the algorithm summarized in [7]

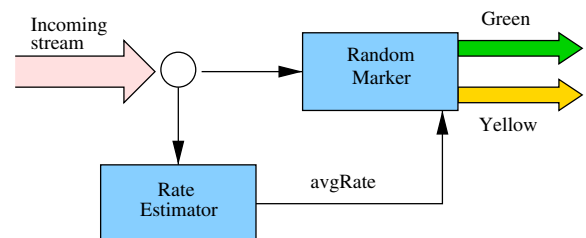


Fig. 4. Scheme of the TSW2CM.

in order to provide an estimate of the running average bandwidth. It takes burstiness into account and smooths out its estimate to approximate the long-term measured sending rate of the traffic stream. The marker uses the estimated rate to probabilistically associate packets with one of the two colors. Using a probabilistic function in the marker is beneficial to TCP flows as it reduces the likelihood of dropping multiple packets within a TCP window.

As a matter of fact, the literature reports other choices of aggregate markers such as [6,11,21]; however, we picked the TSW2CM due to its simplicity and to its TCP-oriented design. Indeed, Token Bucket based markers [5,6] do not add randomness to the marking choice, which makes them less appealing when dealing with an aggregate stream of TCP flows.

Since the Rate Estimator is time-based, its bandwidth estimation depends on the last window, i.e., on the amount of history (recent time) that is used by the algorithm for estimating the rate. Therefore the size of the estimation window considerably influences the identification of fast variations of the ingress rate. The smaller its value, the shorter the time needed to detect rate variations. The value of 1 s has been successfully used for aggregates of TCP flows (as demonstrated in [7]), therefore we set this value.

### 3.1. DS servers and BE clients

In the single-bottleneck simulation scenario, we compare the CT required to complete a Web transaction when an SLA is protecting only server flows, and client requests use the standard BE class of service.

Fig. 5 shows in the top plot the absolute values of the CT versus the DS offered load  $\alpha_{DS}$ . Dotted lines plot the CT when only BE class of service can be used, while the solid lines plot the CT when servers are protected by the SLA. As can be seen, the protection offered by the introduction of DS marking greatly enhances the performance of Web-like flows, both in underprovision and overprovision regions. On the same figure, we report the CT averaged over all flow sizes, and the one obtained for shorter and longer replies, respec-

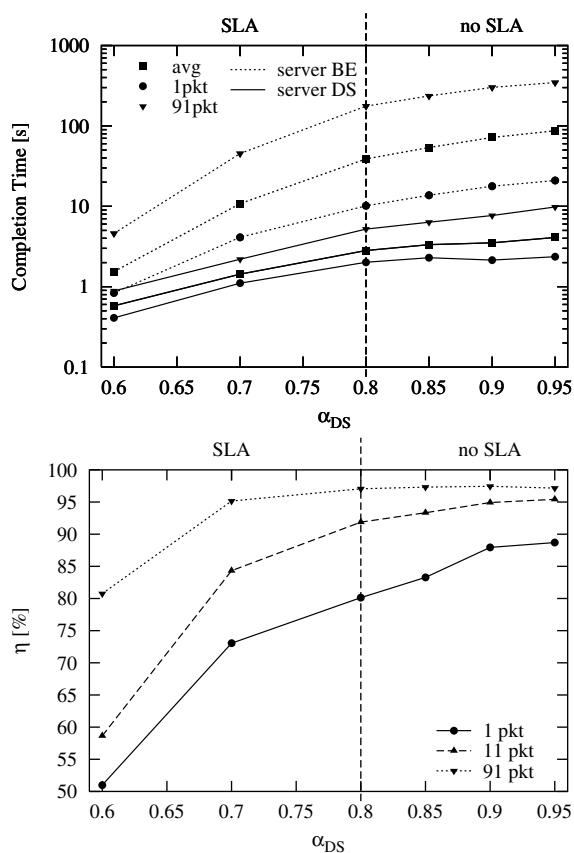


Fig. 5. Completion times of HTTP transactions under DS provisioning of servers only: absolute (top) and relative (bottom) values.

tively 1- and 91-packets-long. Also in this case, there is no surprise in observing that all flows obtain a good performance increase when DS traffic is used to transport server replies.

### 3.2. DS servers and DS clients

The time spent by the server to complete replies is obviously the largest component in the CT. But also the time spent to open a TCP connection and to send the client request may be relatively high, especially when the path from the client to the server is congested. Also, the loss of acknowledgment segments may negatively impact TCP operations. We ran another set of simulations in which clients too are protected by a TSW2CM marker to evaluate the impact of having an SLA also at the client

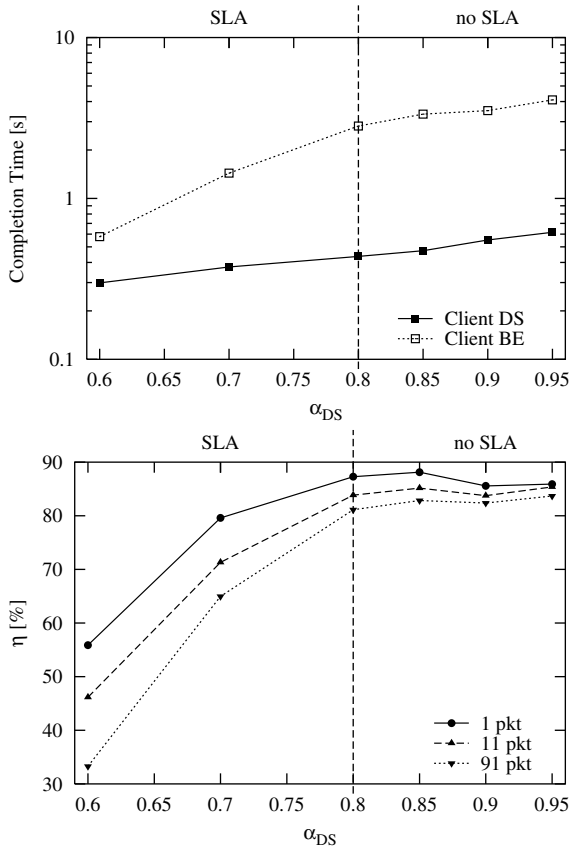


Fig. 6. Completion times of HTTP transactions under DS provisioning of servers and clients: absolute (top) and relative (bottom) values.

side. Given the much smaller quantity of bytes sent from the client to the server, only a 560 kbit/s CIR is required to guarantee 80% of client traffic. The top plot in Fig. 6 shows the average CT when clients send BE only segments (dotted line) or have subscribed to an SLA (solid line).<sup>1</sup> As can be seen, the presence of a small SLA at the client side can reduce the CT by about one more order of magnitude when compared to the case in which only servers are protected. Furthermore, also in the underprovision region we observe great performance improvements, and the average completion time is always smaller than 0.6s, a value that pro-

vides the average Web-browsing user the perception of real-time operations, especially when compared to an average completion time of 4s required under the same traffic conditions (but only with server traffic protected).

This result underscores the importance of providing a minimum protection to client traffic (both data and signaling) so as to better exploit DS capabilities.

### 3.3. Relative performance

To clearly identify the performance increase, the bottom plot of Fig. 5 reports the relative gain

$$\eta = 100 \cdot \frac{CT(BE) - CT(DS)}{CT(BE)}$$

comparing BE flows to DS ones, for flows 1-, 11- and 91-packets long. We observe that a large performance increase is obtained for all flows, but we notice that the flows that better exploit the DS protection are the ones that have to send the largest amount of data. This situation has already been pointed out in [13–15], and is a direct consequence of using a simple TSW2CM, which is unable to discriminate packets belonging to “TCP fragile flows”, i.e., flows whose TCP congestion window is small. The intuition behind this is that TCP suffers when not enough acknowledgments are received, so that segment drops can only be detected when the Retransmission Timer (RTO) fires, and not by the Fast Recovery/Fast Retransmit mechanisms. The impact of this on the CT is very large, being the RTO much longer than a RTT. On top of that, a new Slow Start phase follows a RTO, resulting in the congestion window being small again.

Moreover, the larger enhancement obtained by long-lived flow is a direct consequence of the initial three-way handshake and slow start phase adopted by the TCP protocol too, which delay and slow down the real data transfer. This initial transient period affects short file transfers much more than long-lived flows.

When the aggregate flows of server replies exceed the CIR (underprovision region), not all packets can be marked as *green* by the TSW2CM. Among the different flows that enter the marker,

<sup>1</sup> We omit the 1-, 11- and 91-packets-long figure for the sake of clarity.

the ones that stand bigger chances of being marked in-profile are the longer ones, as they manage to send packets at a higher rate (what with the larger congestion window). This makes it even harder for shorter flows to take advantage of the DS protection, and this is also the reason why longer flows perform relatively better than shorter ones.

The bottom plot of Fig. 6 similarly reports  $\eta$  when comparing the cases in which clients have DS protection or not. In this case minor differences can be noticed among flows, as both short and long flows receive about the same relative performance enhancements, although shorter flows have a clear edge over longer ones. This behavior can be explained by the impact that the latency due to connection opening and initial slow-start phase has upon short and long flows: in the latter case protecting initial segments of clients affects the total completion time in a relatively minor way, hence the smaller gain. To wit, recall that the penalty of timeout expiration upon the loss of a SYN segment is set to 3 s [22].

Considering the difficulties that DS shows in giving equal improvements to long- and short-lived TCP flows, we design a new *Fair Marker* (FM), that tries to overcome all previously under-scored problems, without requiring per-flow management at edge nodes.

#### 4. Fair markers architecture

The design guidelines of the Fair Markers follow those stated in previous works [12,14,15]:

- detect when a TCP connection is in Slow Start, and mark as *Green* the first few segments of the connection;
- detect when a TCP connection is performing a retransmission, and mark as *Green* the retransmitted segment as well as the first few segments following it.

We will refer to the situation just outlined as “critical state”. The practical effect of marking the first packets of a flow is to notify the network that it is too soon to signal the new flow about ongoing congestion: the benefit to the network

would be marginal, what with the small window size of that connection, while the flow would be excessively penalized from the onset.

Of course, the simple inspection of a TCP segment does not allow the marker to infer for how long the flow has been active, or whether or not it is in one of the critical states, therefore some form of connection tracking is required. A possible solution of the problem is to keep track of all active flows across the network and to partition the available DS bandwidth among them so that in-profile flows in critical states are *given priority* [12,14,15]. However, while this is feasible for a small number of flows, it becomes unpractical in most of today’s Internet scenarios. The solution we chose is to limit the number of flows for which state is kept, trying to identify a small set of flows that have been most recently active.

In Fig. 7, the general architecture of the Fair Marker is sketched: the operations of each block are the following:

- *Flow Classifier*: has the function to update the list of active flows, within memory constraints;
- *Pre-Marker*: passes the packets of a flow to the Random-Markers pre-marking them;
- *Rate Estimator*: estimates the rate of the packets at the ingress of the Random-Marker by using a moving-window average;
- *Random-Marker*: possibly re-marks the packets by comparing the average rate with the CIR, in order to keep the rate of the ingress packets within the SLA.

Most listed blocks with the exception of the Flow Classifier can be found in standard DiffServ

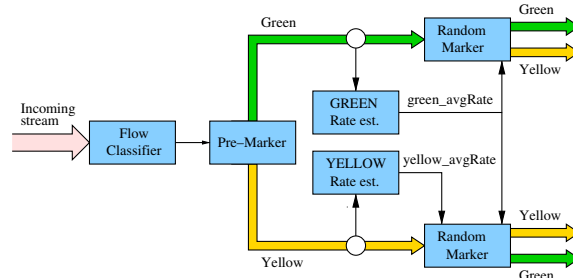


Fig. 7. Architecture of the fair marker.

architectures. The contribution of this paper lies in the use we make of these standard blocks and in their interaction with the operations of the Flow Classifier, described below. As in standard DS architecture, the marked packet stream enters a queue managed by an active queue management algorithm.

#### 4.1. The Flow Classifier

The Flow Classifier shortlists the flows that have been recently backlogged, tracking their activity in terms of number of packets sent. The core of the Flow Classifier is a data structure called *Short-Range Flow Table* (SRFT), which is a classic Least-Recently-Used (LRU) Cache. The aim of the SRFT is to supply a snapshot of the most recently active flows across a node. Its structure is a list collecting IDs of the flows that most recently have sent at least a packet. For each flow, the SRFT stores the following entries:

- `fid`: flow identifier (i.e., IP source/destination address, TCP source/destination port, IP protocol identifier);
- `num_pkt`: number of packets that have been sent by the flow.

Such information is essential to the operations of the Classifier and of downstream blocks, as will be described below. The table size, i.e., the maximum number of tracked flows, is limited to `SRFT_size`.

When a new connection is opened, or when it resumes transmission after a lengthy silent spell due to an RTO, the flow data are entered into in the SRFT. As will be described later, this insertion ensures that the flow receives a preferential treatment, that will help it leave the small window regime more promptly. The finite size of the SRFT ensures that the amount of tracked flows does not become unmanageable. The SRFT is arranged in chronological order, so that, at opposite ends of the table, the most recently active and the least recently active flows can be found. Specifically, new entries are inserted in tail, while the detection of new packets belonging to already active (and listed) connections, force the corresponding entry

to be moved back to tail, so as to preserve the chronological order of packet arrival. For this reason, there is no need to keep track of the time elapsed since the flow was backlogged.

When a flow ID is popped out of the table from the head, the information regarding the amount of data the flow has been sending is lost. Therefore, as far as the marker is concerned, the next time a packet of that flow is detected, its new insertion will be interpreted as that of a new flow, all previous state information being lost. Obviously, a long-lived flow that is evicted from the FM table will get the privilege of a short-lived flow (i.e., marked green) upon re-entering the table. While this behavior is not strictly what the marker is aiming at, it cannot be completely avoided when operations are based on partial state information.

The size of the table `SRFT_size` is, quite obviously, a critical parameter that affects the performance of the Fair Marker. Indeed, too large a table slows down marker operations by forcing it to operate on a per-flow basis; furthermore, it does not allow the identification of time gaps between retransmissions. Conversely, a smaller table size results in near-stateless operations, due to flows constantly popping in and out of the table, and losing state information in the process. In Section 7 we will describe an analytical methodology to dimension it and we will observe the impact of the SRFT table size on completion times.

Detailed operations of the Flow Classifier are described in Fig. 8. When the Classifier has matched the information in IP/TCP headers with the *flow identifier* (`fid`) stored in the SRFT, it

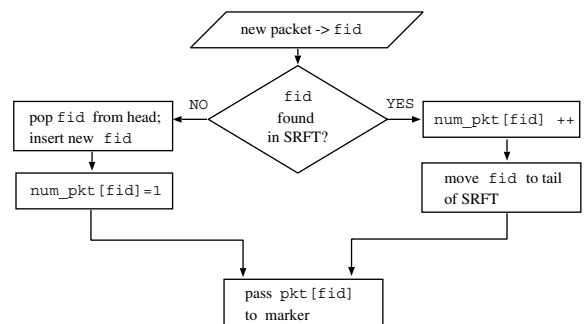


Fig. 8. Flow Classifier operations.

updates the number of packets already sent by that flow and signals such number to the next block that will handle the incoming packet, i.e., the Marker. If no match is found, the Classifier pops out the oldest flow ID, creates a new entry in the SRFT, and signals the Marker that the current packet belongs to a newly listed flow.

Summarizing, the SRFT provides two needful services to downstream blocks:

- identification of recently-active flows;
- bookkeeping of amount of data sent per flow.

#### 4.2. The pre-marker

The aim of the Pre-Marker is to decide how to mark the incoming packet according to the information received by the Flow Classifier. Specifically, it compares the number of packets already sent by the flow with the value of a threshold  $t_{ft}$ . If the number of sent packets is smaller than the threshold, it is an indication that the flow to which the packet belongs may be in a critical state: the packet is therefore pre-marked *Green* and forwarded to the *Green* Random-Marker, (which may in turn decide upon a different re-marking, as will be detailed below). Otherwise, if the number of sent packets is larger than the threshold, it is an indication that the flow has likely left the critical state and no longer needs protection: the packet is then pre-marked as *Yellow* and forwarded to the *Yellow* Random-Marker, where its color may be changed. The aim of these operations is to give lower priority to packets of longer-lasting flows.

Clearly, the operation of this block strictly depends upon the value of  $t_{ft}$ ; in principle, such value can be adapted to variations of the bit-rate of the ingress aggregate flow, although we did not investigate the solution any further. Indeed, properly setting the threshold, although potentially beneficial, has a relatively minor impact on the level of utilization of the network. It should be pointed out that marking segments of longer-lasting flows as *Yellow* (simply because these flows have already sent too many segments with respect to the threshold  $t_{ft}$ ) does not lead to needless discarding of segments under low load conditions. Indeed, at low

loads, the average size of the RED virtual queues at the bottleneck router will be so small as to avoid any probabilistic discarding even of *Yellow* segments.

#### 4.3. The rate estimator

It is a classic Rate Estimator, as described in [7] providing an estimate of the traffic stream arrival rate at its ingress. This rate should approximate the average bandwidth consumption for the traffic stream passed by the Pre-Marker to the Random-Marker over a specific period of time. In our architecture, we distinguish between a *Green* and a *Yellow* Rate Estimators, depending on the branch on which they operate.

#### 4.4. The random-marker

The Random-Marker uses the CIR information and the average rate estimated by the corresponding Rate Estimators to decide how to re-mark incoming packets before forwarding them to the virtual queue. The *Green* Random-Marker has the task of keeping the average rate of *Green* packets under the portion of the CIR that is dedicated to the corresponding type of traffic. Its operations are sketched in the top plot of Fig. 9. The *Yellow* Random-Marker checks whether the *Green* average rate is smaller than the CIR (which means that the SLA is underprovisioned), and, if so, probabilistically remarks *Yellow* packets to *Green* to fill the portion of bandwidth that the *Green* Random-Marker left unused, as shown in the bottom plot of Fig. 9. The principle upon which these Random-Markers operate is the same as that of TSW2CM.

The goal of the two Rate Estimators and Random-Markers is to ensure that the rate of *Green* packets always remain within the boundaries of the subscribed SLA, while at the same time maximizing the SLA bandwidth utilization.

## 5. Simulation results

Using the simulation scenarios outlined in Section 2, in which both clients and servers exploit

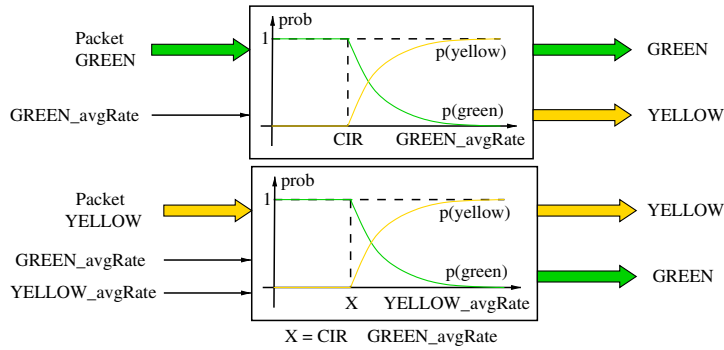


Fig. 9. Green (top) and Yellow (bottom) random-marker operations.

DiffServ-reserved bandwidth, we have repeated the previous experiments comparing the TSW2CM with the FM scheme proposed above. In the performance evaluation we also include simulation results comparing the effects of adopting Early Congestion Notification mechanism (ECN) [23], and briefly discuss its impact on the Completion Time. The reason of introducing ECN in this study is to assess the performance of the marking scheme in a possible more controlled network, in which packet drop are limited by the introduction of Early Congestion Notification mechanism that improve TCP performance, and observe if there is any need of introducing advanced marking mechanism.

The tunable parameters of the FM were chosen as follows:  $t_{ft} = 7$  and  $SRFT\_size = 30$ . In Section 7, we will provide a few results that justify the choice.

The top plot of Fig. 10 depicts the average completion time of HTTP transactions, comparing FM (solid line) TSW2CM (small-dot line). TCP-NewReno sources are used, coupled with (square point) or without (triangle point) ECN marking. As can be seen, the FM outperforms the TSW2CM, especially when ECN is not used. Notice that the largest improvement is achieved when the offered load exceeds the subscribed SLA, in which the effectiveness of the FM becomes even more noticeable. In order to gain greater insight about the performance of flows of different size in comparison to TSW2CM, the bottom plot of Fig. 10 shows the relative gain

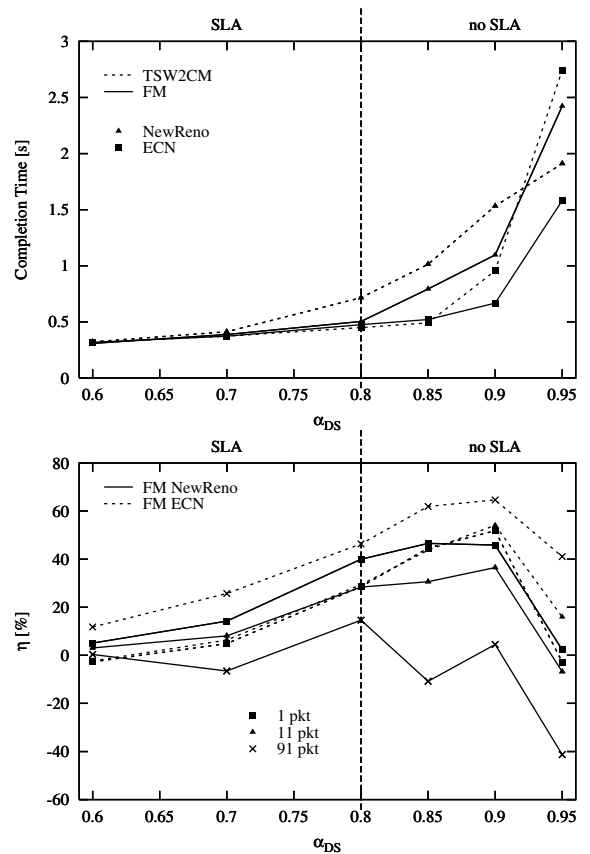


Fig. 10. Completion times of HTTP transactions using different markers and considering a TCP-NewReno source with or without ECN: absolute (top) and relative (bottom) values.

$$\eta = 100 \cdot \frac{CT(TSW2CM) - CT(FM)}{CT(TSW2CM)}$$

for flows whose server reply sizes are 1-, 11- or 91-packet long. Considering the scenario in which ECN is not used, it is important to observe that the goal of improving the level of protection of shorter flows, hence the fairness, is achieved, as shown by the gain of 1- and 11-packet long flows. Furthermore, the impact on longer flows is negligible (the gain  $\eta$  is close to 0). On the contrary, when ECN is used, the effectiveness of the Early Congestion Notification is larger for longer flows, while short-lived flows are unable to take advantage of the improved congestion control. This is not surprising, since ECN cannot kick in for flows that last only few segments.

To give the reader more details, Fig. 11 plots on the top the relative gain of TCP-NewReno sources

with ECN over sources without ECN, when a TWS2CM marker is used. Again, we observe that the largest improvements obtained by introducing ECN are exploited by longer-lived flows, while shorter-lived flows obtain worse performances in the underprovisioning region. Bottom plots of Fig. 11 reports the same relative gain, but in a scenario in which the FM marker is used. It can be observed that the per-flow marking benefits from the use of ECN and, while longer-lived flows exhibit the largest performance improvements, shorter-lived flows also obtain large gains. However, when the network load is very high (95%), we observe that ECN no longer is as effective as before. This is due to the increased dropping probability that arises when the ECN mechanism fails in controlling the TCP rate.

In summary, the improvements shown in the plots are due to the FM ability of selecting packets belonging to flows in “critical states” and offering them the protection they need in order to speed up the completion of their transfers.

### 5.1. Simulations with multi-bottleneck topology

Results for the multi-bottleneck scenario, shown in Fig. 12, confirm the gains of our Fair Marker in terms of completion times. We report results only for HTTP transactions activated between the DS Server 0 sources and the corresponding Client 0 destinations, whose traffic crosses several bottlenecks in the presence of cross traffic. Also in this case, the FM marker is capable of improving the completion times of short-lived flows. Also, the performance of UDP flows, competing for bandwidth on the multi-bottleneck topology, are not affected by the use of either marker, as shown in Fig. 13, which reports the average delay and jitter experienced by UDP segments versus the competing portion of DS traffic  $\alpha_{DS}$  on the top and bottom plot respectively. As expected, no differences are observed when comparing the performance obtained adoption a TSW2CM or the FM, because, being the UDP traffic always backlogged, the pre-marker does not kick in; only a random marking is therefore used, perfectly mimicking the behavior of the classic TSW2CM.

Comparing the performance obtained between DS or BE CBR flows, we observe that the green

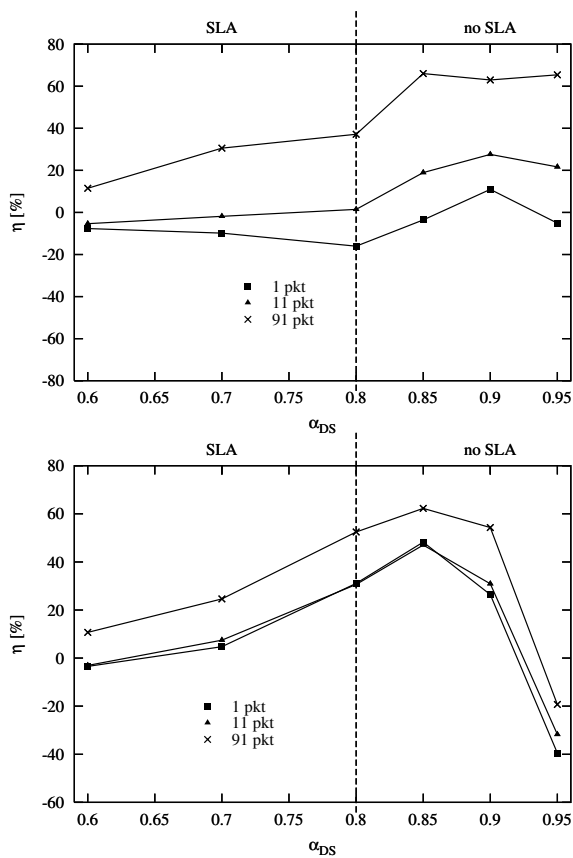


Fig. 11. Relative completion times of HTTP transactions using TSW2CM marker (top) and FM3 marker (bottom) when considering TCP-NewReno with ECN versus TCP-NewReno without ECN.

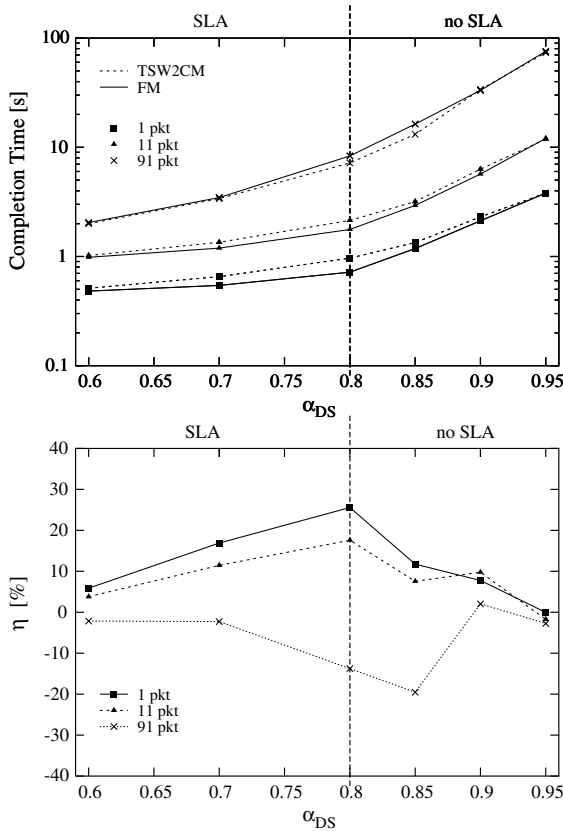


Fig. 12. Completion times of HTTP transactions using different markers: absolute (top) and relative (bottom) values. Multi-bottleneck topology.

CBR source exhibits increasing average delay and almost constant jitter when  $\alpha_{DS}$  increases. This is justified by the increased congestion in the green traffic, which causes larger but constant queuing delay. On the contrary, the yellow CBR source exhibits a decreasing average delay, but much large jitter. Indeed, the yellow traffic suffers for much severe congestion, which causes larger jitter and higher dropping probability. This causes an apparent decrease in the average delay because packets are successfully delivered to the recipients during periods of non-congestion.

## 6. Modeling TCP flows in DiffServ networks

In this section, we describe a TCP model that can be used to predict the performance obtained

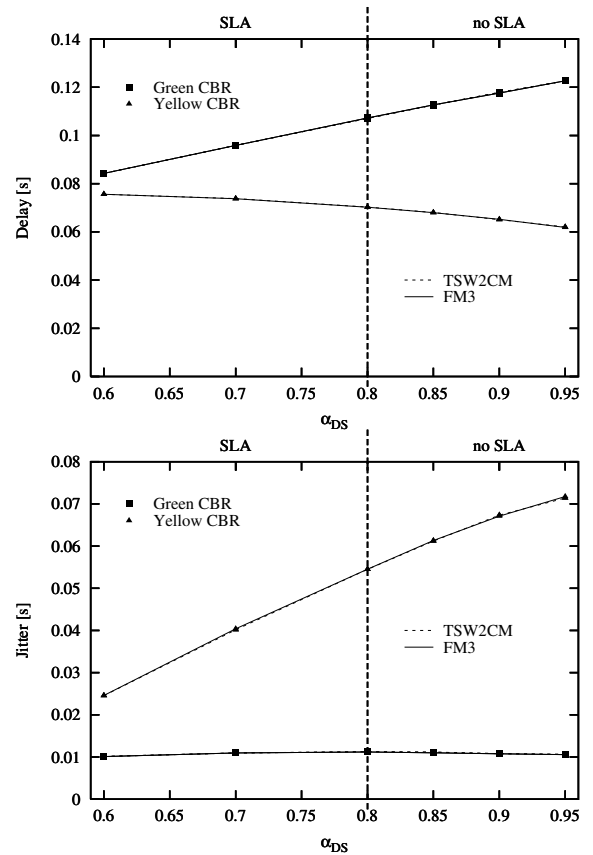


Fig. 13. Delay (on the top) and Jitter (on the bottom) of UDP flows with different markers. Multi-bottleneck topology.

by a flow of a given size that is carried by a Diff-Serv network. The model we are proposing is an extension of [24,25], which have been extended and simplified to deal with the DiffServ framework, and in particular to deal with TSW2CM and FM markers. Our goal is to estimate the *completion time*. A transfer is considered successful when the source receives the ACK for the last segment (we ignore the closing procedure). We consider two separate cases: (i) short-lived flows, and (ii) long-lived flows. In the first case, we extend the model presented in [24] to deal with the Diff-Serv paradigm, while the second case will be considered by extending the model presented in [25].

We consider the reference Slow Start algorithm implemented in BSD UNIX. Since all major TCP proposals, such as TCP Reno, NewReno, SACK implement the same Slow Start and similar loss

recovery algorithms, our model accurately predicts the average completion times for all these TCP variants. For simplicity, we do not model the delayed acknowledgment mechanism. We assume that the sender is limited only by the congestion window size, and not by the receiver window size.

We assume that all segments of a flow follow the same path and that the loss rate is uniformly distributed, i.e., each segment is dropped with the same probability. Note that the assumption that the loss rate is uniformly distributed is likely to hold in practice for short lived flows. Indeed, in the case of high speed links, the correlation between subsequent losses affecting the same flow is highly reduced by the high statistical multiplexing on the link. Modern buffer management schemes such as RED/MRED further reduce the probability of dropping consecutive segments of the same flow. The average RTT is assumed to be known.

To describe our model we use the following variables, which are supposed to be given:

- $R$ —average RTT;
- $p_{tr}$ —probability of *Yellow* segment to be remarked as *Green*;
- $p$ —*Green* segment dropping probability,  $up = 1 - p$ ;
- $q$ —*Yellow* segment dropping probability,  $uq = 1 - q$ ;
- $k$ —*composed Yellow* segment dropping probability,  $uk = 1 - k$ ;
- $T$ —estimated RTO duration;
- $t_{ft}$ —SRFT threshold;

where  $k$  can be obtained simply by  $k = q(1 - p_{tr}) + pp_{tr}$ .

### 6.1. Modeling short-lived TCP flows

Let us evaluate the average TCP connection setup: when FM is considered and  $t_{ft}$  is at least 1, the average connection setup time  $C_s$  can be expressed as

$$\begin{aligned} C_s^1 &= R + up \sum_{i=1}^{\infty} p^i \sum_{j=1}^i 2^{j-1} T \\ &= R + T \frac{p}{1-2p}, \text{ if } t_{ft} \geq 1. \end{aligned} \quad (1)$$

The same expression, but with  $k$  in place of  $p$ , is valid when  $t_{ft}$  is 0, and models the connection setup in a TSW2CM.

Consider now a data transfer of a flow of a given size. Let  $C_{m,c}^{w,j}$  be the average time spent to successfully send  $m$  segments with an initial congestion window of size  $w$ , with  $c$  credit left to deterministically mark consecutive *Green* segments, but with  $j$  credits available in case of RTO. Similarly, let  $C_{m,c}^1$  be the time needed to successfully open a connection and send  $m$  data segments with  $t_{ft} = c$ .

For example, a 7-segment flow Completion Time, with  $t_{ft}$  equal to 5, is given by<sup>2</sup>

$$C_{7,5}^1 = C_s^1 + C_{1,1}^{1,1} + C_{6,2}^{2,5}.$$

Next we derive  $C_{m,c}^{w,j}$ . If  $m = 1$ , the time needed to successfully send a packet is similar to  $C_s^1$ , but there are three different cases, depending on  $t_{ft}$  value:

$$C_{1,1}^{1,1} = R + up \sum_{i=1}^{\infty} p^i \sum_{j=1}^i 2^{j-1} T = R + T \frac{p}{1-2p},$$

$$C_{1,0}^{1,1} \approx R + \frac{k}{p} up \sum_{i=1}^{\infty} p^i \sum_{j=1}^i 2^{j-1} T = R + T \frac{k}{1-2p},$$

$$C_{1,0}^{1,0} = R + uk \sum_{i=1}^{\infty} k^i \sum_{j=1}^i 2^{j-1} T = R + T \frac{k}{1-2k}.$$

(2)

With a burst of two segments we have six cases, which can be expressed by:

$$\begin{aligned} C_{2,c}^{2,j} &= R(1-P)(1-Q) + (1-P)Q(R+T+C_{1,j}^{1,j}) \\ &\quad + P(1-Q)(T+C_{1,j}^{1,j}) + PQ(T+C_{2,j}^{1,j}) \end{aligned} \quad (3)$$

being

$$P = \begin{cases} p & \text{if } c > 0 \\ k & \text{if } c \leq 0 \end{cases} \quad \text{and} \quad Q = \begin{cases} p & \text{if } c > 1, \\ k & \text{if } c \leq 1. \end{cases}$$

The derivation of  $C_{m,c}^{w,j}$  for other combinations of  $m, w$  can be obtained extending the previous reasoning. The reader interested in the whole derivation can refer to [26]. This procedure can be repeated to evaluate  $C_{m,c}^{w,j}$  for  $m > 9$ , but the complexity grows exponentially due to all the possible

<sup>2</sup> Recall that two *Green* credits, if available, are spent during the three-way handshake.

combinations that must be taken into account. Therefore we prefer to follow the ideas in [25], and complete this model with a steady-state model to estimate the TCP performance of long-lived flows.

### 6.2. Modeling long-lived TCP flows

To evaluate Completion Times for long lived-flows we can refer to [25], in which a new model for TCP performance is presented that extends the steady-state results from [27] by deriving new models for two instances that can dominate TCP latency: the connection establishment and the initial Slow Start. We further extend that model to be able to deal with different dropping probabilities, i.e., considering yellow and green packets. In particular, we assume that  $p \approx 0$ , i.e., no *Green* segments are dropped; this assumption is correct only if the DiffServ network is carefully dimensioned, i.e., the provisioning of the DS bandwidth is well configured [13].

Repeating the same derivation as in [25,27], we report the final expression of the average throughput  $B$  experienced by a long-lived TCP source which is given by:

$$B \approx \min \left( \frac{W_{\max}}{R}, \frac{Q + \frac{1}{k} + Q t_{ft} + \sqrt{\frac{8}{3} \left( \frac{1-k}{k} + Q t_{ft} \right)}}{R \left( 1 + \sqrt{\frac{2}{3} \left( \frac{1-k}{k} + Q t_{ft} \right)} \right) + T} \right) \quad (4)$$

in which  $W_{\max}$  is the maximum sender congestion window,  $Q = \min(1, 3/w)$  is the probability of detecting a loss by a RTO. The Completion Time is then determined by dividing the flow length by the average throughput  $B$ .

### 6.3. Model validation

To cross-check the model results with simulation results, we ran several test cases, which confirm the accuracy of the model [26]. We report here only a selection of those results. Fig. 14 shows the comparison between the proposed model and the simulation. The short-lived flow model was used for flows of length 1 to 9 segments, and it is detailed in the bottom plot. The full picture, including results derived with the long-lived flow model for flows longer than 9 segments, is shown in the top plot. The simulation scenario is composed by flows that have to send a given amount of packets, and cross a buffer which artificially drops packets according to two fixed dropping probabilities. In particular, we set  $p = 10^{-4}$  and  $q = p$ ,  $q = 10p$  and  $q = 100p$ . An FM is present, in which the  $t_{ft}$  threshold has been set to 8. It can be seen that the TCP model very well predicts the Completion Times of short-lived flows, while the long-lived part of the model is less accurate.

Fig. 15 instead reports simulation results of the more complex and realistic scenario we used to derive performance evaluation in Section 5. We ran

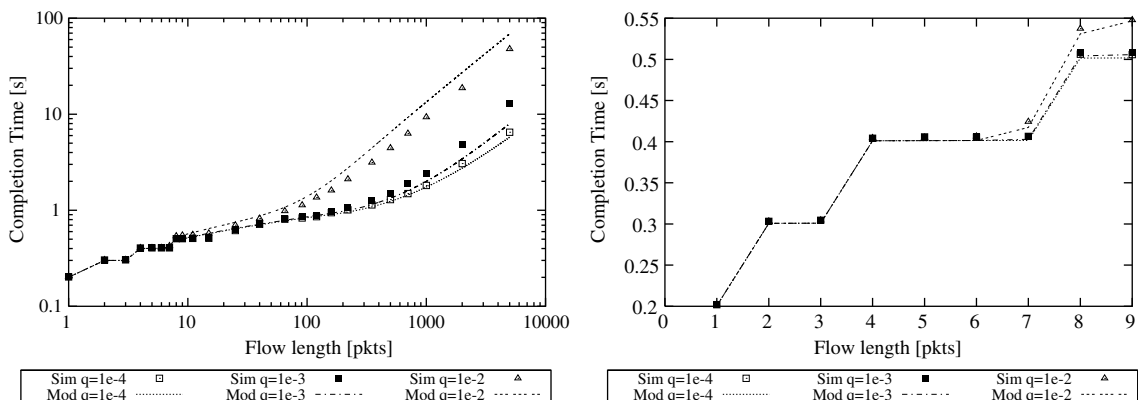


Fig. 14. Completion time for different values of the yellow dropping probability.

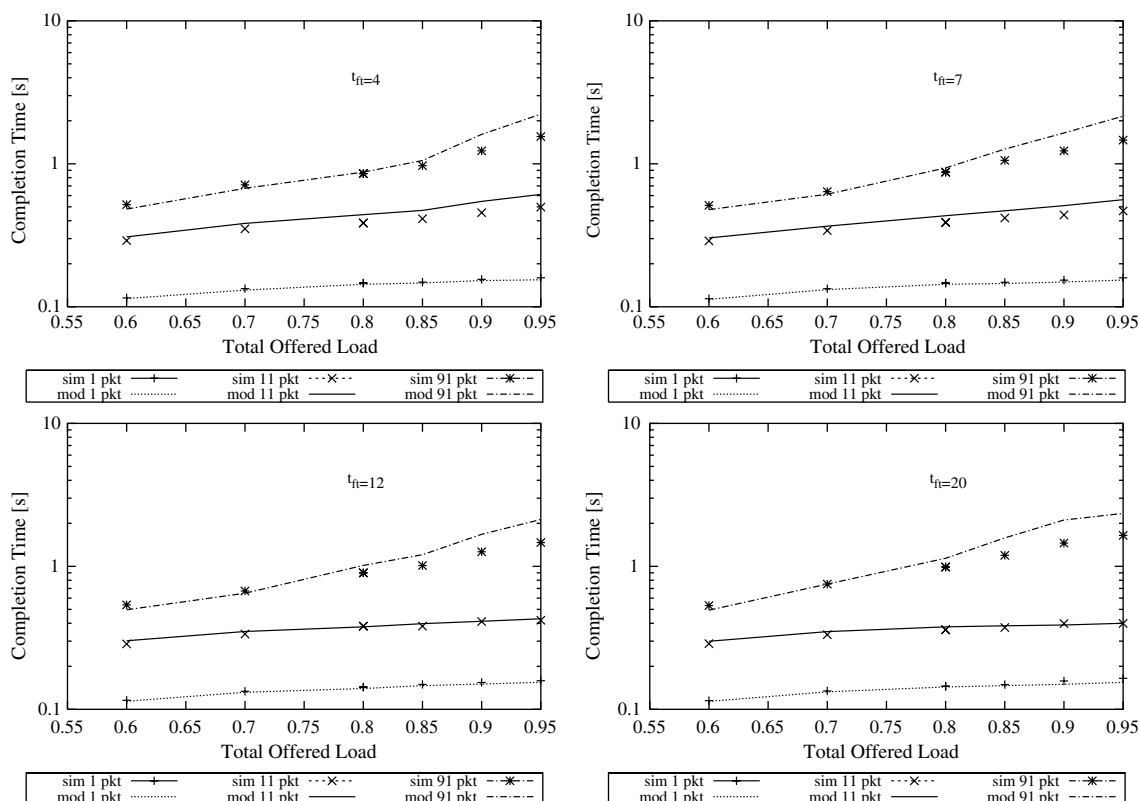


Fig. 15. Comparison between the proposed model and the simulation measurements for different  $t_{ft}$ .

simulations for increasing total offered load to the bottleneck link, and then measured the  $RTT$  and  $q, p$  values which are later used by the model to predict the Completion Time of different flows. In particular, different values of the  $t_{ft}$  threshold were selected, and results are plotted for the 1, 11, 91 packet long flows. Also in this case the model exhibits a very good prediction of the Completion Times, and therefore can be used to predict the performance experienced by users with different configuration of the FM marker.

## 7. Parameter tuning

So far, two parameters,  $t_{ft}$  and  $SREFT\_size$  have been identified as “tunable” parameters. The setting of  $t_{ft}$  is quite simple, since the number of TCP segments that take a connection out of the “critical state” is deterministically known: 7 seg-

ments if delayed acknowledgments are not used, 10 segments otherwise. As such, the proper value of  $t_{ft}$  can safely be set within this restricted interval of values. This intuition is confirmed by simulation: in Fig. 16 (top plot), we varied the range of  $t_{ft}$  under several  $\alpha_{DS}$  in the single-bottleneck scenario. The curves of completion times as a function of  $t_{ft}$  show a minimum in the range between  $t_{ft} = 5$  and  $t_{ft} = 15$ , for all values of  $\alpha_{DS}$ . This justifies our previous choice of  $t_{ft} = 7$ . Given the traffic distribution earlier described, this corresponds to allowing a 1-Mbit/s, i.e., about 15% of the contracted SLA, of pre-marked *Green* traffic by the Marker. The remaining portion of packets are (re-)marked as *Green* by the *Yellow* Random-Marker. For higher values of  $t_{ft}$ , i.e., larger than 15, completion times become longer, as all DS flows have most of their packets pre-marked as *Green* and sent to the *Green* Random-Marker, which at this point re-marks randomly, much like a plain

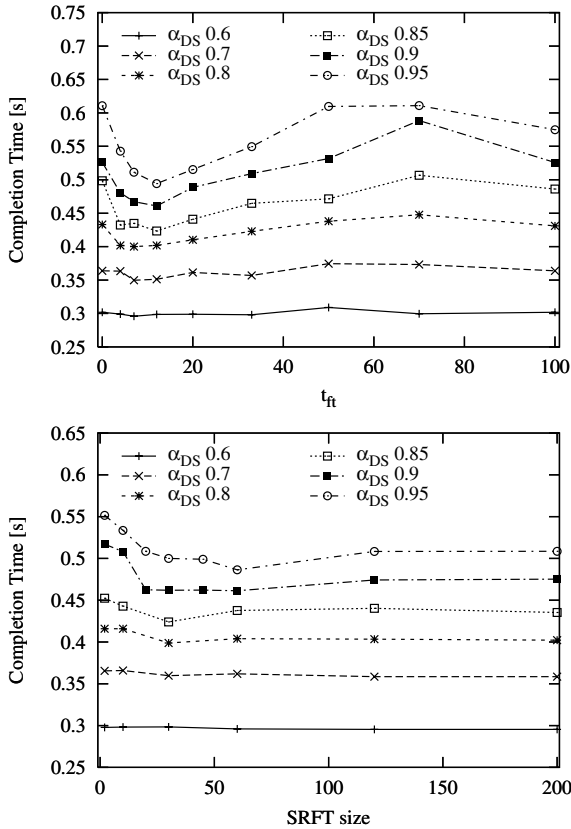


Fig. 16. Parameter tuning: SRFT threshold (top) and SRFT size (bottom).

TSW2CM, in order to keep the marked packet stream within the subscribed CIR.

Conversely, the choice of `SRFT_size` is more complex, although the model outlined in Section 6 can provide a few guidelines. Indeed, assuming that the flow length distribution is known, the Completion Time predicted by the model can be used to evaluate the average number of contemporary active flows on a bottleneck queue. Considering that a bottleneck queue crossed by TCP flows can be modeled by a M/G/1 processor sharing queue, the first step is determining the average service time required to serve a client, i.e., a TCP flow. This quantity is the average Completion Time of flows in the network: given the average Completion Time for each flow length,  $E[C_i]$  (computed from the short-lived flow and long-lived flow models), then the average Completion Time  $E[CT]$  is simply computed as

$$E[CT] = \sum_{i=1}^{\infty} E[C_i]p_i, \quad (5)$$

where  $i$  represents the flow length in segments and  $p_i$  the probability that a flow consists of  $i$  segments.

Stating by  $E[\lambda]$  the average rate at which flows arrive at the queue, from Little's formula, we obtain

$$E[N] = E[\lambda]E[CT]. \quad (6)$$

Eq. (6) can then be used to dimension the *Short-Range Flow Table*, e.g. setting `SRFT_size`= $E[N]$ . Fig. 17 reports some numerical values of  $E[N]$  as a function of  $E[\lambda]$  computed using Eq. (6);  $E[CT]$  was derived by Eq. (5) for the flows length distribution given in Table 2. These results show that setting `SRFT_size` to a value around 20 is a safe choice at high loads.

We investigated the matter further using simulation: the results, in the bottom plot of Fig. 16, confirm the findings of the model. It can be seen that, at low loads, a small value of the `SRFT_size` is enough to get optimal results, as predicted by the average number of active flows for low loads in Fig. 17. At higher loads, a value between 20 and 30 can be safely identified as optimal, while smaller values of the SRFT are unable to correctly keep track of the flow state. Again, this is in accordance with the number of active flows predicted by the model. Notice also that for values larger than 40, we always observe a small increment in the

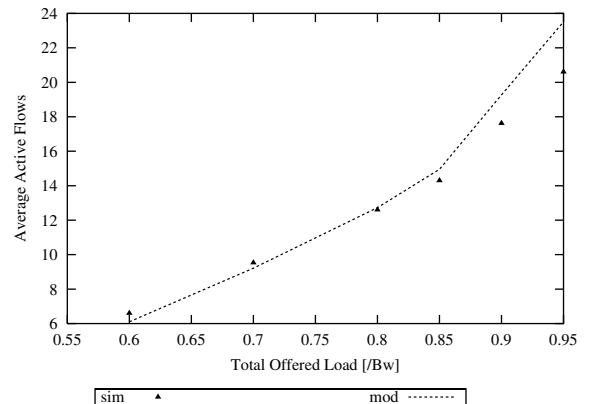


Fig. 17. Number of contemporary active flows measured during simulation and obtained by the model, for different values of the total offered load.

CT, due to the failure of the FM to detect long idle periods that follow RTOs, and consequently to protect the retransmitted segment and the following Slow Start phase. It is also important to notice that the introduction of the FM always improves the CT, for every value of `SRFT_size`. Indeed, if `SRFT_size = 0`, it behaves as a classic TSW2CM. This shows an intrinsic robustness achieved in the design of the FM.

## 8. Conclusions and future work

In this paper we have discussed a lightweight solution to the problem of fair DiffServ protection of both short- and long-lived TCP flows. Our design guidelines have followed the principle of memory-limited, partial state information inside edge nodes, avoiding the scalability drawbacks of per-flow marking and management. In terms of performance, our solution was shown to be as effective as per-flow marking, and to greatly improve with respect to the performance of standard stateless DiffServ markers. The proposed architecture relies on a small set of tunable parameters, whose optimal values were discussed both using simulation and modeling techniques. Adaptive implementations may be devised to automatically adjust those parameters. Finally, we have also addressed the provision of minimum protection to both sides of asymmetric traffic crossing congested domains.

## Acknowledgement

The authors wish to thank Massimo Alzati and Marco Bottigliengo for writing part of the simulation code.

## References

- [1] R. Braden, D. Clark, S. Shenker, Integrated services in the Internet architecture: an overview, RFC 1633, June 1994.
- [2] D. Black, S. Blake, et al., An architecture for differentiated services, RFC 2475, December 1998.
- [3] V. Jacobson, K. Nichols, K. Poduri, An expedited forwarding PHB, RFC 2598, June 1999.
- [4] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, Assured forwarding PHB group, RFC 2597, June 1999.
- [5] J. Heinanen, R. Guerin, A single-rate three-color marker, RFC 2697, September 1999.
- [6] J. Heinanen, R. Guerin, A two-rate three-color marker, RFC 2698, September 1999.
- [7] W. Fang, N. Seddigh, B. Nandy, A time sliding window three color marker, RFC 2859, June 2000.
- [8] S. Sahu, D. Towsley, J. Kurose, Quantitative study of differentiated services for the Internet, IEEE Globecom'99, Rio de Janeiro, BR, December 1999, pp. 1808–1817.
- [9] W. Feng, D. Kandlur, D. Saha, K. Shin, Understanding TCP dynamics in a differentiated services Internet, IEEE/ACM Transactions on Networking 7 (April) (1999) 173–187.
- [10] W. Feng, D. Kandlur, D. Saha, K. Shin, Adaptive packet marking for maintaining end-to-end throughput in a differentiated-services Internet, IEEE/ACM Transactions on Networking 7 (April) (1999) 685–697.
- [11] M. El-Gendy, G. Kang, Equation-based packet marking for assured forwarding services, IEEE Infocom 2002, New York, NY, June 2002, pp. 23–27.
- [12] A. Das, A. Helmy, Fair stateless aggregate marking techniques using active queue management techniques, in: IEEE/IFIP Conference on Management of Multimedia Networks and Services (MMNS), Santa Barbara, CA, October 2002.
- [13] D. Rossi, C. Casetti, M. Mellia, A simulation study of Web Traffic over DiffServ networks, in: IEEE Globecom 2002, Taipei, TW, November 2002, pp. 2585–2589.
- [14] M. Mellia, I. Stoica, H. Zhang, TCP-aware packet marking in networks with diffserv support, *Computer Networks* 42 (1) (2003) 81–100.
- [15] G.L. Monoco, F. Azeem, S. Kalyanaraman, Y. Xia, TCP-friendly marking for scalable best-effort services on the Internet, *Computer Communication Review* 31 (5) (2001).
- [16] W. Nouredine, F. Tobagi, Improving the performance of interactive TCP applications using service differentiation, *Computer Networks* 40 (September) (2002) 19–43.
- [17] M. Alzati, M. Bottigliengo, C. Casetti, M. Mellia, Fair marking of web flows using partial state information, International Teletraffic Congress (ITC-18), Berlin, Germany, September 2003.
- [18] WETMO Web page, Available from <<http://www.tlc-networks.polito.it/wetmo>>, 2004.
- [19] S. Floyd, RED: discussions of setting parameters, Available from <<http://www.icir.org/floyd/REDparameters.txt>>.
- [20] M. Alzati, M. Bottigliengo, Study and simulation of fair markers in DiffServ networks, Master Thesis, Politecnico di Torino, Italy, July 2002, Available from <<http://www.tlc-networks.polito.it/mellia/papers/alzati.ps.gz>>.
- [21] Y. Chait, C.V. Hollot, V. Misra, D. Towsley, H. Zhang, J. Lui, Providing throughput differentiation for TCP flows using adaptive two color marking and multi-level AQM, IEEE Infocom 2002, New York, NY, June 2002, pp. 23–27.
- [22] R. Braden, Requirements for Internet hosts—communication layers, RFC 1122, October 1989.

- [23] K.K. Ramakrishnan, S. Floyd, D. Black, The addition of explicit congestion notification (ECN) to IP, RFC 3168, September 2001.
- [24] M. Mellia, I. Stoica, H. Zhang, TCP model for short lived flows, *IEEE Communications Letters* 6 (2) (2002) 85–87.
- [25] N. Cardwell, S. Savage, T. Anderson, Modeling TCP latency, *IEEE Infocom 2000*, Tel Aviv, IS, March 2000, pp. 367–375.
- [26] M. Alzati, M. Bottigliengo, C. Casetti, M. Mellia, Modeling TCP performance with FM3 and TWS2CM, Technical Report, Available from <<http://www.tlc-networks.polito.it/mellia/papers/TWS2CM.ps>>.
- [27] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling TCP reno performance: a simple model and its empirical validation, *IEEE/ACM Transactions on Networking* 2 (April) (2000) 133–145.



**Claudio Casetti** received his PhD in Electronic Engineering from Politecnico di Torino in 1997. He is an Assistant Professor at the Dipartimento di Elettronica e Telecomunicazioni of Politecnico di Torino. He has coauthored about 70 journal and conference papers in the fields of networking. His interests focus on performance evaluation of TCP/IP networks and wireless communications.



**Marco Mellia** received his degree in Electronic Engineering in 1997, and a PhD in Telecommunications Engineering in 2001, both from Politecnico di Torino. From March to October 1999 he was with the CS department at Carnegie Mellon University as visiting scholar. Since April 2001, he is with the Dipartimento di Elettronica e Telecomunicazioni of Politecnico di Torino as Assistant Professor. He has

co-authored over 50 papers published in international journals and presented in leading international conferences, all of them in the area of telecommunication networks. He participated in the program committees of several conferences including IEEE Globecom and IEEE ICC. His research interests are in the fields of All-Optical Networks, Traffic measurement and modeling, QoS Routing algorithms.