TCP smart framing: a segmentation algorithm to reduce TCP latency

# TCP Smart Framing: A Segmentation Algorithm to Reduce TCP Latency

Marco Mellia, *Member, IEEE*, Michela Meo, *Member, IEEE*, and Claudio Casetti

***Abstract*—*TCP Smart Framing*, or TCP-SF for short, enables the Fast Retransmit/Recovery algorithms even when the congestion window is small. Without modifying the TCP congestion control based on the additive-increase/multiplicative-decrease paradigm, TCP-SF adopts a novel segmentation algorithm: while Classic TCP always tries to send full-sized segments, a TCP-SF source adopts a more flexible segmentation algorithm to try and always have a number of in-flight segments larger than 3 so as to enable Fast Recovery. We motivate this choice by real traffic measurements, which indicate that today's traffic is populated by short-lived flows, whose only means to recover from a packet loss is by triggering a Retransmission Timeout. The key idea of TCP-SF can be implemented on top of any TCP flavor, from Tahoe to SACK, and requires modifications to the server TCP stack only, and can be easily coupled with recent TCP enhancements.**

**The performance of the proposed TCP modification were studied by means of simulations, live measurements and an analytical model. In addition, the analytical model we have devised has a general scope, making it a valid tool for TCP performance evaluation in the small window region. Improvements are remarkable under several buffer management schemes, and maximized by byte-oriented schemes.**

***Index Terms*—Performance evaluation, protocol design, TCP enhancements.**

## I. Introduction and Work Motivation

BALANCING greediness and gentleness has always been the distinctive feature of congestion control in the TCP protocol [1]. Mindful of the presence of other traffic sharing the same network resources, TCP tries to grab as much bandwidth as possible, eventually causing congestion and data loss. Data lost by TCP is used as congestion signal, and cause the source to slow down its transmission rate. Thus, lost data can actually be seen as bandwidth used to control and regulate the network, since every segment the network discards is an indication that a TCP source has been congesting the network and should temporarily back off.

This scheme has been successfully applied over the years, while the traffic pattern has shifted from long file transfers and short, persistent connections, typical of terminal-emulation traffic, to the "Click-and-Run" paradigm found in Web interactions. The design of the segmentation algorithm we are proposing was motivated by the analysis of real traffic measurements that highlighted the prominent role of short-lived flows in today's Internet traffic.

In this paper, we propose a new approach to data segmentation in the early stages of Slow Start that adheres to the TCP guidelines listed in [2] and, at the same time, addresses the nature of today's Internet traffic: short, spotty client-server interactions between a Web client and a Web server. We will refer to this variant of TCP as *TCP Smart Framing*, or TCP-SF for short.

As will be detailed below, we advocate an increase in the number of segments transmitted by a TCP source, without increasing the amount of application data actually sent in the congestion window. This will be done whenever the congestion window is "small", i.e., at the beginning of each Slow Start phase, and in particular at the connection startup.

The main observation is that Classic TCP's congestion control is only marginally driven by the rate at which *bytes* leave the source but, rather, by the rate at which *segments* (and their respective ACKs) are sent (or received) at the source.

TCP infers that a segment is lost whenever one of the following two events occurs: a Retransmission Time Out (RTO) expiration, or the arrival of three duplicate ACKs that triggers the Fast Retransmit (FR) algorithm. Of these two events, RTO is the most undesirable one as the RTO period is usually much larger than the Round Trip Time (RTT).[1] Indeed, regardless of the actual amount of bytes transmitted, a coarse RTO expiration can be prevented only if enough segments are sent in the transmission window (i.e., at least three more following the lost segment). This situation can occur only if: 1) the congestion window is larger than 4 MSS (Maximum Segment Size) and 2) the flow is long enough to allow the transmission of at least 4 segments (i.e., it is not a so-called *short-lived* flow). Also, it should be pointed out that repeatedly forcing a short-lived connection into RTO often results in excessive penalty for the connection itself, that would otherwise be finished in few more segments, rather than in actual network decongestion.

While Classic TCP[2] starts sending one segment, in our scheme a TCP-SF source is allowed to send $N_{\mathrm{sf}}$ segments, whose aggregate payload is equal to the MSS associated to the connection. Thus, the resulting network load is, byte-wise, the same of a Classic TCP connection (except for the protocol overhead). When the window size grows past a threshold, the Classic TCP behavior is restored.

Manuscript received January 7, 2003; revised October 28, 2003; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor V. Padmanabhan. This work was supported by the Italian Ministry for University and Scientific Research under the project PlanetIP and by the Center for Multimedia Radio Communications (CERCOM). A preliminary version of this paper was presented at the IEEE GLOBECOM 2001, San Antonio, TX.

The authors are with the Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy (e-mail: mellia@mail.tlc.polito.it; michela@mail.tlc.polito.it; casetti@mail.tlc.polito.it).

Digital Object Identifier 10.1109/TNET.2005.846901

[1]RTO is at least 1 s, to account for Delayed ACKs and avoid spurious RTO expirations [3].

[2]Unless otherwise specified, by "Classic" TCP we refer to any TCP version currently implemented in standard TCP stacks (i.e., TCP Tahoe [4], TCP Reno [5], TCP NewReno [6], and TCP SACK [7]).

The Classic TCP algorithms (Slow Start, Congestion Avoidance, Fast Retransmit, Fast Recovery) are not otherwise affected. However, the modification introduces a number of key advantages:

- the lengthy first-window RTO (set to 3 s) is no longer the only outcome if a loss occurs at the onset of a connection;
- when Delayed ACKs are employed and the congestion window is 1 segment large, the receiver does not have to wait up to 500 ms before generating an ACK; several current TCP implementations, start a connection with a window of 2 segments, a widely employed, acknowledged workaround to the Delayed ACK initial slowdown;
- short-lived flows, for which the Completion Time is paramount, are less likely to experience a coarse RTO expiration, since the number of transmitted segments grants a bigger chance of triggering FR;
- shorter segments can exploit pipelined transmission, completing the transfer in a shorter time because of the store-and-forward at routers; this is especially useful in slow links;
- in a wireless environment, shorter packets suffer smaller transmission error probability;
- not requiring any contribution from the receiver, the scheme can quite easily be deployed on a source-only basis; furthermore, it can equally benefit well-established Classic TCP flavors, such as TCP Reno, NewReno, SACK, and also works coupled with Early Congestion Notification (ECN).

In the rest of the paper, we will first provide some measurement results that motivate the development of TCP-SF (Section II), then we will describe in detail the TCP-SF algorithm (Section III), a combined simulation/measurement performance analysis (Section IV) and an analytic study of the the performance of our scheme (Section V). In Sections VI and VII, we compute and evaluate the impact of buffer management and protocol overhead introduced by TCP-SF. Section VIII will conclude the paper.

## II. Traffic Measurement as Motivation

In order to motivate our work, in this section we report some measurements derived from real traffic traces. In particular, we present measurement results that show:

- the impact of congestion control on the Completion Time experienced by TCP flows;
- the file size distribution of TCP connections, highlighting the importance of this problem.

The measured flow length distribution will also be used to provide a realistic scenario in the performance analysis section.

We use the measurement testbed based on the Tstat tool [8], [9] that was developed at the Politecnico di Torino. Tstat is a passive measurement tool which, among other capabilities, tracks the status of single TCP connections throughout their life.

During the roughly three years since the first working version of Tstat was available, several traces collected at our Institution gateway were parsed. These data are collected on the Internet access link of Politecnico di Torino, i.e., between the border router of Politecnico and the ingress router of GARR/B-TEN [10], the Italian and European Research network. Within the Politecnico Campus LAN, there are approximately 7 000 hosts; most of them are clients, but several servers are regularly accessed from the outside. The backbone of our Campus LAN is based on a switched Fast Ethernet infrastructure. There is a single point of access to the GARR/B-TEN network and, through it, to the public Internet. Therefore, measurements collected on this access link allow Tstat to track the status of TCP flows, as both forward segments and backward ACKs are observed.

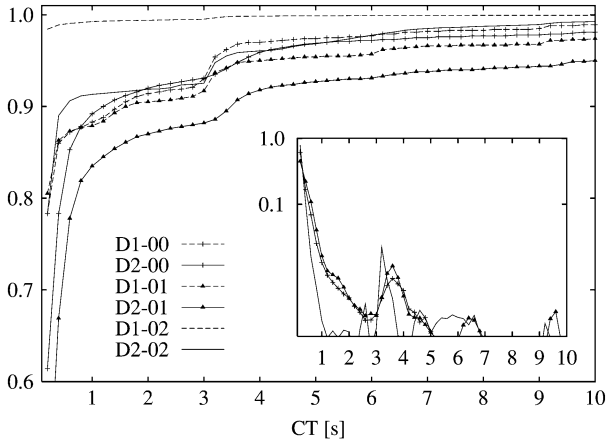



Fig. 1. Server Side Completion Time distribution (CDF and pdf) of TCP flows during different periods.

Among the available data, we selected three periods, in three different years (in brackets, the number of valid TCP flows tracked during each measurement campaign):

- 2000: from 4/11/2000 to 4/14/2000 (3 373 812);
- 2001: from 2/1/2001 to 2/12/2001 (14 038 045);
- 2002: from 10/22/2002 to 10/31/2002 (42 262 892).

The capacity of the access link changed during the measurement periods, from 8 Mb/s in 2000, to 16 Mb/s in 2001, and to 28 Mb/s in 2002.

### A. Server Completion Time

Among the measurements Tstat produces, we look at the *Server Completion Time*—server CT, which is defined, for each TCP connection, as the time elapsed since the first until the last payload-carrying segment coming from the server side is observed, i.e., the time required to receive all the data from the server, neglecting the connection setup, any possible client requests, and the connection tear-down.

Several factors impact on the server CT: 1) the flow data length; 2) the capacity of the path bottleneck; 3) the TCP protocol congestion control and capability of dealing with packet losses; 4) the server latency; and 5) the client latency in generating ACKs. To gauge the impact of the these components, we analyzed the server CT and looked for indications that would allow us to pinpoint the role of each one. Fig. 1 reports the Cumulative Distribution Function (CDF), and the corresponding Probability Density Function (pdf) in the inset of the server CT measured considering selected Internet hosts. In particular, it reports results relative to the two most visited web sites from clients in the campus LAN (identified by $D1$ and $D2$ for privacy), in all three years.[3] As can be observed from the CDF plot, about 85%–90% of TCP connections successfully deliver all data in less than one second, while 3%–7% of connections

[3]Results are similar considering other servers.

TABLE I
COMPLETION TIME (CT) AND PERCENTAGE OF FR PER FLOW CLASS

| No. | Bytes | Classic | | VS-TCP-SF | | FS-TCP-SF | |
|---|---|---|---|---|---|---|---|
| | | CT [s] | % FR | CT [s] | % FR | CT [s] | % FR |
| 1 | 119 | 0.27 | — | 0.27 | — | 0.28 | — |
| 2 | 179 | 0.31 | — | 0.30 | — | 0.32 | — |
| 3 | 251 | 0.36 | — | 0.34 | — | 0.36 | — |
| 4 | 334 | 0.41 | — | 0.39 | — | 0.39 | — |
| 5 | 428 | 0.46 | — | 0.45 | — | 0.47 | — |
| 6 | 529 | 0.53 | — | 0.50 | — | 0.52 | — |
| 7 | 658 | 0.59 | — | 0.58 | — | 0.61 | — |
| 8 | 948 | 0.77 | — | 0.66 | — | 0.69 | — |
| 9 | 1650 | 1.34 | — | 0.92 | 1.3 | 0.96 | 0.9 |
| 10 | 2861 | 1.38 | — | 1.04 | 11.5 | 1.08 | 20.2 |
| 11 | 4706 | 2.04 | — | 1.33 | 9.5 | 1.33 | 34.7 |
| 12 | 8015 | 2.32 | — | 1.57 | 18.0 | 1.54 | 48.5 |
| 13 | 13681 | 2.72 | 19.5 | 1.89 | 30.0 | 1.91 | 55.1 |
| 14 | 26641 | 3.63 | 27.5 | 2.76 | 36.2 | 2.55 | 55.0 |
| 15 | 284454 | 13.92 | 60.3 | 14.46 | 69.3 | 12.22 | 80.5 |

require 3–4 s to complete their transfer, and finally a long tail can be observed due to the heavy-tailed distribution of the payload.

The reason there is a high fraction of TCP connections that require more than 3 s to complete the transfer is due to the expiration of the RTO used by TCP to identify a packet loss. Indeed, looking at the pdf reported in the inset, we clearly identify a quasidiscrete distribution centered close to the RTO and multiple RTO expiration values: 1.5, 3, 6, 9, etc., seconds.

No major differences can be observed between the three different measurement periods, except that from 2001 to 2002, the path to $D1$ changed, turning it into an underloaded, uncongested path. Therefore, most data transfers are accomplished in a very short time. This also suggests that the CT is not dominated by the server latency, but by the network capacity and TCP mechanisms. Nonetheless, it is still possible to notice a small step in the CDF around 3–4 s.

### B. Flow Length

Tstat also derives the *flow length*, i.e., the byte-wise size of the payload transported on each direction of the full-duplex TCP connection. In particular, we are interested in the distribution of the amount of data servers sent to clients. It is well known that most of the current Internet traffic is built by short-lived flows (see for example [11]), due to the predominance of HTTP traffic. And, it is well known that the flow size distribution exhibits a heavy-tailed distribution [12]. This also suggests that for most of the flows, the only way to recover from a packet loss is by triggering the RTO.

From the measured flow length, we derive a discretized distribution by splitting the flows in 15 groups with the same number of flows per group, and then computing the average flow length in each group. The results of the 2002 measurement can be found in columns 1 and 2 of Table I. As can be seen, more than 80% of the flows are shorter than 14 Kbytes (corresponding to less than 10 full-size segments, by taking today's most common MSS of 1460 bytes as measuring unit[4]) for which FR can not be triggered. Moreover, more than 50% of flows account for one data segment only.

[4]In past measurement works found in the literature, an MSS equal to 536 was very common. Several current measurements, such as the ones obtainable by Tstat [9] or CAIDA [13], place the most common MSS at 1460.

Interestingly, no major differences in the flow distribution were observed during the three measurement periods, while a general increase of the weight of the tail of data payload size was noticed in 2002. This underlines that, while the number of long-lived flows is increasing, the majority of data connections is still carried by short-lived flows, for which the impact of the TCP RTO cannot be neglected.

In the performance evaluation in Section IV, we present results obtained by using the same flow length distribution derived from the latest period of measurements, and discretized according to the 1460 byte-long MSS.

## III. TCP SMART FRAMING

When the TCP congestion window size (cwnd) is smaller than four segments, TCP has no other means to recover segment losses than by RTO expiration. Being the time to recover a loss by RTO expiration much longer than the time needed by FR, this behavior deteriorates the TCP performance, especially when connections are short-lived. In particular, when the flow length is shorter than 10 full-sized segments (i.e., about 14 Kbytes using a 1460-bytes MSS), and the receiver is implementing Delayed ACKs, there are no chances for the transmitter to trigger a FR. If the delayed-ACK option is not implemented, the flow must last more than 7 segments for FR to be triggered.

In the algorithm we are proposing, we enhance the TCP behavior in the operating region where RTO is the only way to recover losses (i.e., when $\mathrm{cwnd} < 4\mathrm{MSS}$) making FR possible, e.g., at the beginning of each Slow Start phase. The region in which we enhance the TCP behavior is commonly known as the *small window regime*.

TCP-SF is based on the following idea: increasing the upstream flow of ACKs by sending downstream a larger number of segments whose size is smaller than the MSS. While maintaining unchanged the amount of data injected into the network, a larger number of segments received at the other end triggers a larger number of ACKs on the backward channel, hence a larger probability that the transmitter can recover losses without waiting for the RTO to expire. In other words, this procedure gives the transmitter the chance to obtain more feedback about what is happening at the receiver. Increasing the number of ACKs will therefore:

- enable FR when the congestion window is smaller than four segments; in particular any flow larger than $3(\mathrm{MSS}/N_{\mathrm{sf}})$ may benefit from this;
- help the RTT estimation algorithm to converge quickly to the correct value of the RTO, thus alleviating the initial RTO penalty of 3 s.

We now illustrate our approach by means of an example, using $N_{\mathrm{sf}} = 4$. Upon the onset of a connection, the congestion window size is equal to one segment, i.e., MSS bytes. If this segment is lost, the transmitter gets no information back, and waits for the RTO to expire before sending the segment again. Now if, instead of sending one MSS-byte-long segment, the transmitter sends four segments whose size is $\mathrm{MSS}/4$, the loss of the first segment can be recovered by FR after the reception of three duplicated ACKs.

Since the enhancement introduced by TCP-SF is needed when only RTO can be used to detect segment losses, the smart framing option is activated when the window size is smaller than the threshold $\mathrm{cwnd}_{\mathrm{min}}$, and Classic TCP behavior (i.e.,

segment size equal to MSS) is switched back as soon as the congestion window is large enough to enable FR. Of course, this behavior applies both at connection start and whenever the congestion window is shrunk to a size smaller than $\mathrm{cwnd}_{\min}$. Given that the FR algorithm is triggered by the reception of three duplicate ACKs, we suggest that $\mathrm{cwnd}_{\min}$ be set to 4MSS.

Let us now elaborate a bit on the small-segment option. Let us define $\mathrm{cwnd}_0$ as the initial congestion window size in bytes,[5] $\mathrm{MSS}_c$ as the value of the maximum segment size at a specific time, while using MSS to identify the "standard" maximum segment size as defined from the knowledge of connection setup parameters. We consider two possible behaviors:

- **Fixed-Size (FS-) TCP-SF**. When $\mathrm{cwnd} < \mathrm{cwnd}_{\min}$, the maximum segments size in use at the sender, i.e., $\mathrm{MSS}_c$, is equal to $\mathrm{MSS}(\mathrm{cwnd}_0/\mathrm{cwnd}_{\min})$; otherwise, $\mathrm{MSS}_c = \mathrm{MSS}$.
- **Variable-Size (VS-) TCP-SF**. The initial segment size $\mathrm{MSS}_c$ is set to $\mathrm{MSS}(\mathrm{cwnd}_0/\mathrm{cwnd}_{\min})$; then, $\mathrm{MSS}_c$ is increased by a factor $\alpha$ after every in-sequence ACK, until the segment size is equal to MSS. Of course, there can be countless choices for $\alpha$, however, we picked the value that yields a segment size equal to MSS for the 9th segment: this choice allows all four segments in the third window (which is 4MSS wide) to have the same size as in Classic TCP. Thus, with $\mathrm{cwnd}_0 = 1\mathrm{MSS}, \mathrm{cwnd}_{\min} = 4\mathrm{MSS}$, in the first window $(\mathrm{cwnd} = 1\mathrm{MSS})$, four segments of size $\mathrm{MSS}/4$ are sent; in the second window $(\mathrm{cwnd} = 2\mathrm{MSS})$, the transmitted segments have sizes $\alpha \cdot \mathrm{MSS}/4, \alpha^2 \cdot \mathrm{MSS}/4, \alpha^3 \cdot \mathrm{MSS}/4, \alpha^4 \cdot \mathrm{MSS}/4$, respectively; the next ACK allows the transmission of the 9th segment, whose size is $\alpha^5 \cdot \mathrm{MSS}/4$. Consequently, the value of $\alpha$ can be determined by imposing that the 9th segment be equal in size to MSS, that is to say $\alpha^5 \cdot \mathrm{MSS}/4 = \mathrm{MSS}$, hence $\alpha^5 = 4; \alpha \approx 1.32$.

To summarize, when cwnd is equal to 2 MSS, Classic TCP sends two segments whose size is equal to MSS bytes. FS-TCP-SF sends eight segments whose size is $\mathrm{MSS}/4$ while VS-TCP-SF sends four segments of variable size. As soon as cwnd reaches the threshold $\mathrm{cwnd}_{\min} = 4\mathrm{MSS}$, the behavior of TCP is no longer critical (RTO expiration ceases to be the only way to recover losses) and the three versions of TCP behave in the same way.

Notice that both TCP-SF versions require a simple modification to Nagle's Algorithm [5], which forbids to send more than one segment shorter than the MSS per RTT. By introducing the notion of $\mathrm{MSS}_c$, Nagle's Algorithm must forbid the sender to inject into the network more than one segment shorter than $\mathrm{MSS}_c$.

One advantage of using TCP-SF with fixed-size segments relies in its simplicity: only two segment sizes are possible, either MSS or $\mathrm{MSS}(\mathrm{cwnd}_0/\mathrm{cwnd}_{\min})$. However, the overhead introduced increases with cwnd. On the contrary, when using variable-size segments, TCP-SF keeps the overhead constant but a more careful implementation is required to deal with variable-size segments.

Let us point out and summarize some critical issues related to the implementation of TCP-SF.

[5]We consider a TCP implementation where initial window (IW) and loss window (LW), as defined in [1], take the same value.

- The degree of aggressiveness of TCP-SF is the same as other classical versions of TCP, at least in terms of bytes sent. In fact, the evolution of cwnd as well as the amount of data submitted to the network are unchanged.
- The proposed enhancement can be applied to any version of TCP, since they all adopt the same mechanism to detect segment drops. Moreover, it is suitable to be used coupled with an ECN-enabled network. Also even if delayed-ACK is enabled, TCP-SF will trigger FR because the receiver disables the delayed-ACK feature when out of order segments are received.
- The implementation of TCP-SF is extremely simple. It requires to slightly modify the transmitter behavior while maintaining the receiver unchanged. This modification translates into a few lines of code in the TCP stack.
- The main disadvantage is that TCP-SF increases the overhead of a factor equal to the segment size reduction factor; i.e., using four segments per MSS, the TCP-SF overhead is four times the Classic TCP overhead. This issue is discussed in Section VII.

### A. Related Proposals

Similar proposals that address the RTO penalty can be found in the literature. A feature they share with TCP-SF is that they only require modifications of the sender's TCP stack.

TCP Limited Transmit [14] allows the transmission of *new* segments upon the reception of duplicate ACKs, so as to enhance the chances of triggering FR if the window is smaller than four segments at the time of packet loss. It should be noted that, compared to our proposal, TCP Limited Transmit cannot trigger FR for flows shorter than 5 MSS; also, at least one more RTT is required before FR can be entered.

An increase of the initial cwnd is advocated in [15] and [16], through an increase in the permitted upper bound for the initial window from one segment to between two and four segments, without changing the segment size; this approach, already implemented in some OSs, can help avoid RTOs in the initial window, but could potentially increase the network congestion and, acting more aggressively, it could affect other TCP sources not employing this algorithm, as the authors themselves point out. Coupled with the Limited Transmit proposal, an initial window larger than two enables the FR for flows longer than 4 MSS.

Allowing TCP to reduce, in certain special circumstances, the number of duplicate acknowledgments required to trigger a fast retransmission, is among the latest proposals [17].

Another potentially related TCP modification proposal, called Appropriate Byte Counting (ABC) is detailed in [18]. It proposes that the congestion window be increased depending on the number of previously unacknowledged bytes covered by each ACK, rather than on the number of received ACKs. Specifically, it dictates that a TCP sender in Slow Start increases the congestion window by the number of previously unacknowledged bytes ACKed by each incoming acknowledgment, provided the increase is not more than $L$ bytes, with $1 \cdot \mathrm{MSS} \leq L \leq 2 \cdot \mathrm{MSS}$. While preventing bursty congestion window increases caused by largely cumulative ACKs, ABC can shorten the Completion Time by avoiding the effect of delayed ACKs. This proposal is also shown to slightly increase the dropping rate at routers [18].

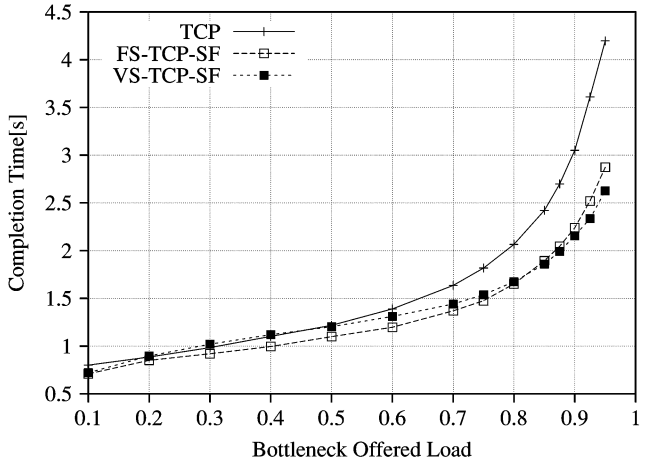



Fig. 2. Average Completion Time.

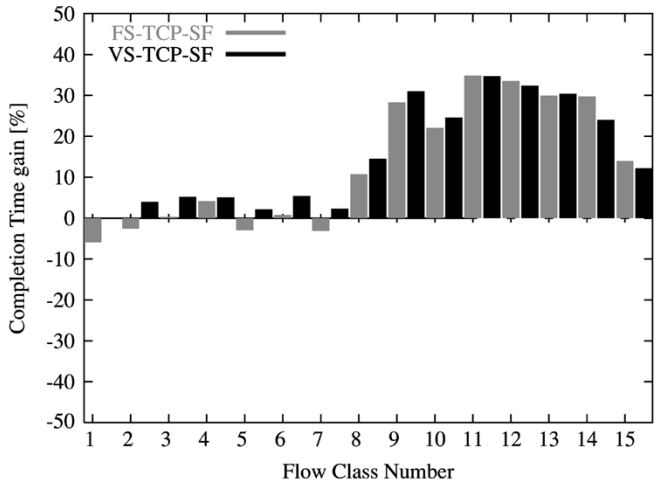



Fig. 3. Per-flow-length Completion Time gain at 80% offered load.

All these proposals, including TCP-SF, are not mutually exclusive, but rather can be merged in a single, new TCP protocol stack that better suits the current Internet traffic characteristics.

## IV. Performance Evaluation

We have chosen to investigate the performance of TCP-SF using both simulation and actual testbed measurements. Simulation gives us full control over specific scenarios, allowing us to derive several performance indices. On the other hand, the testbed implementation allows a more realistic evaluation featuring actual traffic patterns.

We implemented both flavors of TCP-SF in the ns-2 simulator [19]. For our testbed measurements, we implemented the Fixed-Size version in the Linux kernel 2.2.17.

### A. Simulation Results

We report results for a network scenario in which both long-lived FTP sessions and Web-like connections share a common bottleneck link.

To model Web-like traffic, TCP sources (and receivers) are connected to a bottleneck link of 10 Mbit/s capacity and 50 ms delay. A Poisson process drives the setup of new connections, whose length (in bytes) is randomly set according to the real traffic distribution shown in Section II. In the preliminary versions of this paper [20], we presented simulation results with flow-length distributions taken from [11], yielding similar conclusions. The MSS is set to 1 460 bytes. Long-lived connections are accounted for using 10 greedy TCP sources. The bottleneck link is managed by a byte-wise Drop-Tail buffer, whose capacity is set to 150 Kbytes, i.e., 100 full-size packets. Results are averaged over 8 independent simulation runs, and, for each run, the simulation time is 4000 s to reduce the impact of the initial transient phase.

*1) TCP-SF Performance Evaluation:* First, we focus on the performance evaluation of TCP-SF versus Classic TCP-SACK, considering the Completion Time, i.e., the time that is required by the source to successfully transfer all the data, as the most important metric.

Fig. 2 reports the Completion Time versus the average offered load normalized with respect to the bottleneck capacity. Results are averaged over all classes of flows. As expected, the average Completion Time required to complete the transfer is larger when the offered load grows. This is due to two main reasons: first, the average window size shrinks as the capacity of the link is shared among a larger number of flows. Second, the probability of segment loss is larger, causing either RTO or FR to occur. These two effects force the source to operate in the small window region, where FR is unlikely to be triggered, and thus the RTO is the only way to recover from segment loss.

Both VS- and FS-TCP-SF outperform Classic TCP, as they are quicker to react to segment losses. In particular, when the offered load is smaller than 80%, FS-TCP-SF performs better than VS-TCP-SF. When, instead, the offered load is larger than 80%, the larger overhead introduced by FS-TCP-SF increases the actual offered load and consequently the dropping probability, which degrades the Completion Time. The larger the offered load, the higher the performance improvement observed, which shows that TCP-SF is most effective under more critical network conditions.

We now shift our focus to the more interesting, high-load region in the previous experiment, namely when the offered load is set to 80% of the bottleneck capacity. Fig. 3 reports the percentage Completion Time gain for each flow length class. The percentage gain is defined as the relative difference in Completion Time experienced by a given version of TCP and Classic TCP. The classes are identified by a number; Table I relates them to their respective flow size in bytes.

As can be observed, the majority of classes benefits from the enabling of the segmentation algorithm of TCP-SF even if the number of bytes they have to transmit is insufficient to trigger the Fast Recovery algorithm. Instead, for classes of flows sending enough data to trigger the Fast Recovery algorithm, the Completion Time gain experienced is consistent, and over 30%. Notice that also the longest-lived flows (195-segment long) obtain considerable advantages from the introduction of TCP-SF segmentation, which kicks in every time the flow enters the small window regime. Small negative gains are observed for flows that send a single, undersized segment, using the fixed-size version. These gains are due to the increased overhead carried by the network when the fixed-size version of TCP-SF is used.

In order to explore higher-order statistics, Fig. 4 shows the cumulative distribution function of completion times, at 80% offered load, averaged over all flow sizes, for both TCP-SF and

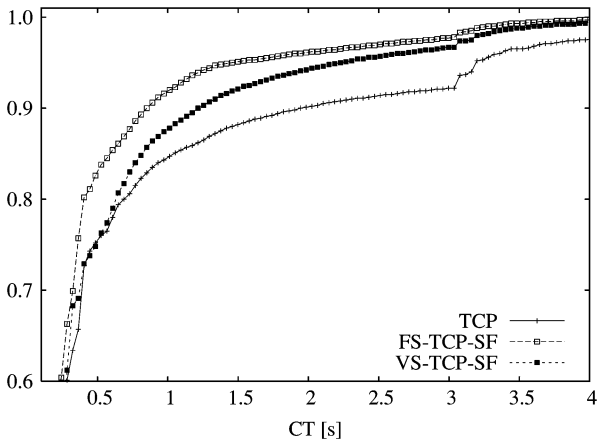



Fig. 4. Distribution of Completion Times at 80% offered load.

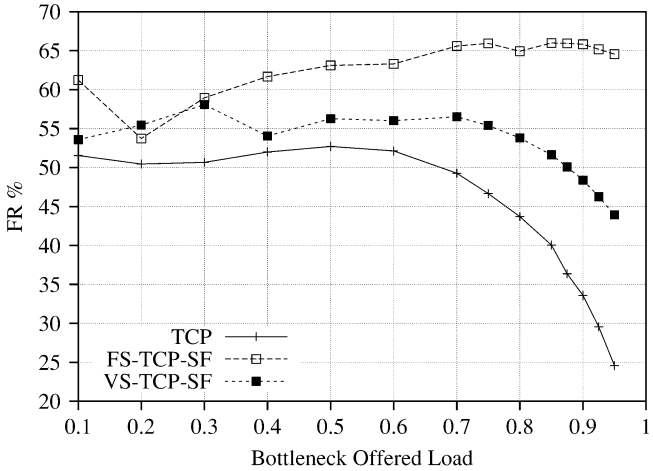



Fig. 5. Percentage of FR events: comparison between TCP-SF and Classic TCP.

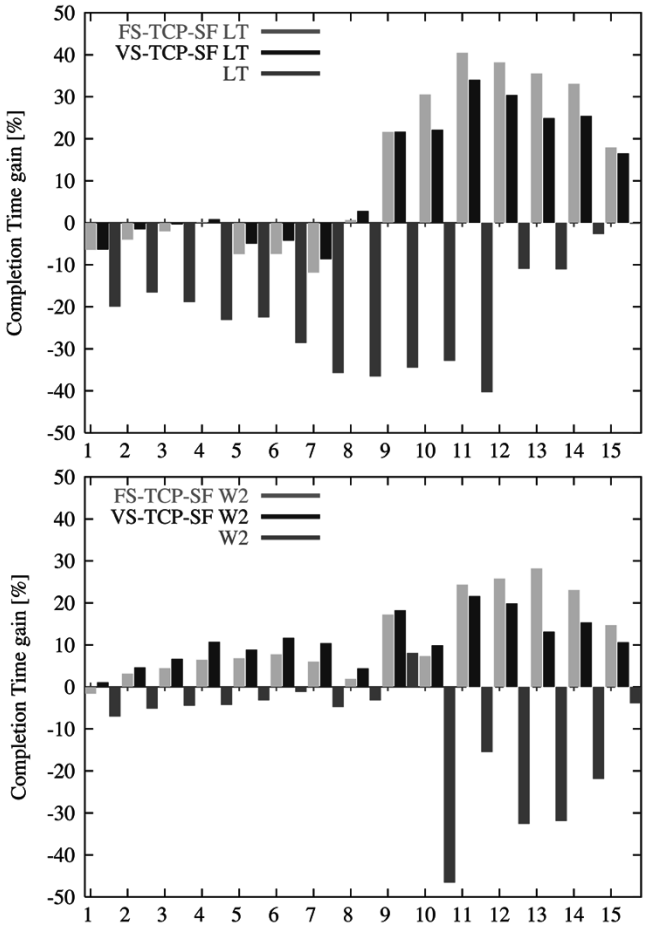



Fig. 6. Per-flow-length Completion Time gain of new TCP proposals.

Classic TCP, confirming the predominance of small completion times in either versions of TCP-SF. In particular, notice the much smaller step in the CDF at about 3 s when TCP-SF is considered. As already remarked in Fig. 1, the step is due to the 3-s RTO expiration whose impact is almost negligible when TCP-SF is used.

To give further insight into the performance that both flavors of TCP-SF obtain, Fig. 5 plots the percentage of times a Fast Recovery is used to recover from a packet loss versus the offered load to the bottleneck link. Notice that Classic TCP recovers from a packet loss using FR about 50% of the time under light load conditions, while this percentage quickly drops to less than 25% when the offered load increases. This causes an increase of the time required to complete the transfer shown by previous plots. The same degradation can be observed considering VS-TCP-SF, which however triggers Fast Recovery for about 45% of drops. FS-TCP-SF exhibits a different behavior, in which the percentage of Fast Recovery is increasing. However, this does not correspond to a proportional decrement in the Completion Time (see Fig. 2) because of the increased dropping probability.

Table I reports, for all classes of flows numbered 1 through 15, the Completion Time and percentage of FR instances used to detect segment losses for Classic TCP, VS- and FS-TCP-SF. For some flow lengths FR is not observed since there are not enough data to trigger it. Specifically, different Completion Times are observed, even if the RTO and FR percentage are the same, because each scenario exhibits a slightly different loss probability.

Table I underlines the benefits obtained by TCP-SF in increasing the number of FR, while at the same time reducing the RTO occurrences. The benefits are clearly visible starting from flows that have 1650 bytes to send. On the contrary, the first class of flows that can use FR for the Classic TCP is the one that has 13 681 bytes to send, for which only less than 20% of dropped segments trigger FR.

*2) Comparison to Related Proposals:* The performance of TCP-SF is compared against two similar TCP proposals described in Subsection III-A, namely Limited Transmit (LT) and TCP with increased initial window (W2). Results obtained by enabling different proposals are shown in Fig. 6. While LT and W2 alone do not always provide gain in Completion Time with respect to Classic TCP, their combination with either flavor of TCP-SF yields enhanced performance. In particular, when LT is adopted, flows shorter than 6 segments, (for which the only means to recover a packet loss is by RTO) are penalized by a 20%–40%. Indeed, the LT option only delays the RTO firing by one or more RTTs. This is due to the fact that TCP adopts a single timer which is associated to the first sent-but-not-yet-acknowledged segment: every received ACK delays this timer by a least one RTT. On the contrary, the W2 option marginally degrades the Completion Time for 1-segment flows; while improving the performance for the 2-segment flows, it degrades

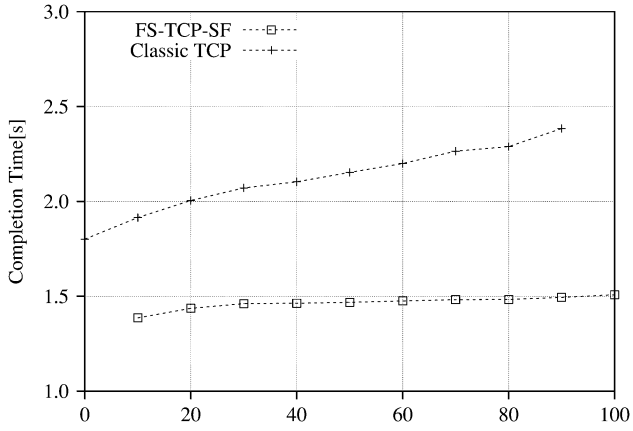



Fig. 7. Completion Time of Classic TCP and Fs-TCP-SF versus different percentage of TCP-SF present in the network.

the Completion Time of flows longer than 2 segments. This is due to the more bursty traffic injected into the network, which causes a higher dropping probability (this is also observed by the authors of the W2 proposal).

These results seem to suggest that rather than seeing these proposals as competitors, they could be merged in a single, new TCP protocol stack.

*3) Friendliness Versus Classic TCP Versions:* One important aspect that must be taken into account when a new protocol is introduced in a heterogeneous networking environment is the impact that it might have on traffic carried by already existing protocols: during the transient phase both new and old versions coexist in the network and the introduction of the new protocol should not penalize previously existing ones. This issue is usually stated in the literature as "friendliness problem."

Thus, we set up a simulation scenario where a variable percentage of TCP-SF sources compete in a bottleneck link with the classic version of TCP. In this experiment, TCP Reno flows load the bottleneck link. The number of TCP-SF flow percentage goes from 0%, i.e., all the sources use Classic TCP, to 100%, i.e., all the sources implement the TCP-SF algorithm, either in the FS or VS option. The normalized load feeding the bottleneck is, as usual, 80% in all simulations.

Fig. 7 reports the experiment results for FS-TCP-SF (results for VS-TCP-SF are omitted for the sake of brevity). The presence of TCP-SF flows degrades the performance of Classic TCP flows. This is due to the fact that TCP-SF better utilizes the buffer space in Drop-Tail queues, causing a slight bias in the dropping probability against full-size packets. In particular, the VS-TCP-SF is friendlier, as it increases the segment size instead of sending a larger number of smaller-size segments.

### B. Testbed Measurements: Real Network, Synthetic Traffic

To gauge the impact of our new scheme in a real Internet environment, we have implemented FS-TCP-SF on a Linux workstation acting as file server. We have then tested the implementation using several clients (adopting different OSs) which requested 10-Kbytes file uploads to the server, the hosts being located both on the local network (in Italy) and in the Internet at large (in the U.S.). Each client repeated the requests some 20 000 times over a time span of 4 hours during peak business hours in Italy.

TABLE II
PERCENTAGE OF FR AND RTO OCCURRENCES MEASURED PER HOST

| Host | Classic | | FS-TCP-SF | |
|---|---|---|---|---|
| | % FR | % RTO | % FR | % RTO |
| sigcomm.cmcl.cs.cmu.edu | 24.2 | 75.8 | 53.2 | 46.8 |
| manchester-1.cmcl.cs.cmu.edu | 11.1 | 88.9 | 72.5 | 27.5 |
| ssh.workspot.net | 19.0 | 81.0 | 72.3 | 27.7 |
| wavelan24.cs.ucla.edu | 34.3 | 65.7 | 69.3 | 30.7 |

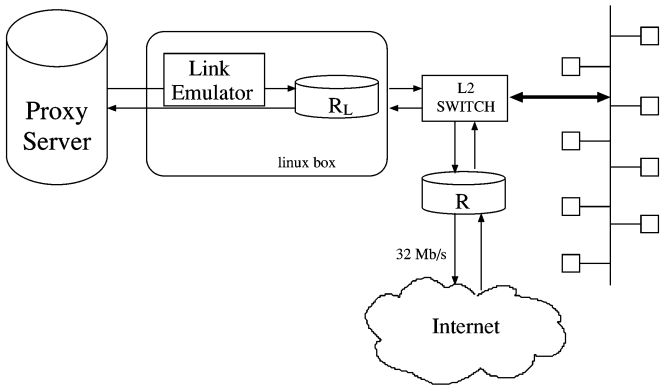



Fig. 8. Topological description of the real traffic experiment.

Results in Table II are reported in terms of FR and RTO occurrences, similarly to those in Table I. In our testbed configuration, the dropping probability is small, since the Italy-to-U.S. path is usually lightly loaded, most of the traffic flowing in the opposite direction. Thus, a small fraction of the flows suffered from segment losses. This is reflected in an average Completion Time measurement that is almost constant, and we opted not to report it. On the contrary, the frequency of FR and RTO occurrences is a more significant metric because it is a relative measure.

On the whole, the testbed results confirm what we observed in simulation: TCP-SF connections resort to FR about four times more frequently than Classic TCP does.

### C. Testbed Measurements: Real Traffic, Synthetic Network

The second set of testbed experiments involved a realistic traffic pattern (i.e., Web client-server transactions) routed over a link emulator. The topology of the testbed is described in Fig. 8. Its functional elements are the following:

- a switched 100 Mb/s LAN carrying Department traffic;
- a Web Proxy Server used by clients;
- a Router $R$ connecting local traffic to the Internet, and carrying outbound and inbound traffic over a 28 Mb/s link;
- a Router $R_L$ implemented on a Linux machine which also runs a Link Emulator, affecting only traffic *from* the Proxy Server;
- all links except for the outgoing ones are Full-Duplex Fast Ethernet links.

In order to generate realistic traffic, every Web Browser in our Department Subnet was configured so as to use a Proxy Server. TCP-SF was only implemented over the machine running the Proxy Server, and a link emulator was added on the return path between the Proxy and the Department Subnet. The link emulator was configured so as to enforce a specific latency and byte-wise drop probability (i.e., longer packets have a higher

TABLE III
RT ESTIMATES, PERCENTAGE OF RTO OCCURRENCES AND COMPLETION TIME DIFFERENCES FOR WEB OBJECTS SMALLER THAN 10 kB VERSUS DROP PROBABILITY AND LATENCY

| drop probability | latency [ms] | Classic TCP Samples | Classic TCP RT [ms] | Classic TCP % RTO | TCP-SF Samples | TCP-SF RT [ms] | TCP-SF % RTO | $\Delta CT_{loss}$ [s] |
|---|---|---|---|---|---|---|---|---|
| 0.01 | 20 | 8093 | 435 | 63.2 | 9743 | 307 | 17.5 | 8.32 |
| | 50 | 3222 | 667 | 64.7 | 3520 | 392 | 15.0 | 14.24 |
| | 100 | 6009 | 824 | 51.3 | 3605 | 681 | 13.5 | -0.25 |
| 0.05 | 20 | 6999 | 482 | 81.4 | 7464 | 333 | 27.8 | 5.63 |
| | 50 | 4994 | 669 | 63.9 | 3406 | 429 | 20.3 | 5.11 |
| | 100 | 7058 | 1206 | 59.7 | 8227 | 652 | 16.0 | 3.48 |
| 0.10 | 20 | 5022 | 477 | 80.5 | 3302 | 397 | 24.0 | 4.23 |
| | 50 | 2859 | 725 | 79.7 | 4601 | 428 | 22.4 | 4.63 |
| | 100 | 1535 | 1008 | 69.5 | 3011 | 661 | 22.3 | 3.80 |

probability of being dropped than smaller ones). The above configuration allows the Web objects fetched by the Proxy and returned to the requesting clients on the Department Subnet to be sent over the link emulator using TCP-SF at the transport layer.

Performance metrics were collected for different values of emulated latency and drop probability, over a period of one month in June 2001; in order to collect a meaningful set of data, each latency/drop pair was set for a whole day, and the Proxy Server had its transport layer switched between Classic TCP and TCP-SF every five minutes. Only connections between the Proxy server and a local client were considered. Statistics were later collected for each version of TCP, and for each day. Unlike the results in the previous subsection, we had little control over the actual client sessions: the amount of data transferred during each transaction depended on the browser used and the operating system installed on each user's machine. Also, each browser has its own session-level behavior, including idle time between back-to-back requests, or connection shut-down after a timeout. Unfortunately, this prevented us from obtaining reliable estimates of Completion Times. Indeed, Completion Times are computed at the Proxy Server as the time elapsed between the reception of a SYN segment from a client, until the last FIN (or RST) segment is received by the server, or sent by the server, whichever occurred last. Flows that did not transport any payload are not taken into account. We have thus chosen to report just the *difference* between the Completion Times of TCP-SF and Classic TCP, assuming that, on the whole, whatever discrepancies between the behaviors of different clients' OSs and browsers would equally affect the two versions of TCP being switched on the Proxy Server. We are aware that these statistics are hardly reliable, but we chose to report them anyway, since they might give an indication as to which version ensures a faster completion.

Also, the percentages of Timeout expiration are reported, as in Subsection IV-B. These statistics were collected by logging actual RTO expiration events of the proxy kernel. Unlike Completion Times, statistics on the occurrence of RTOs and Fast Recovery are less dependent on browser behavior, and are, therefore, more reliable metrics. These results are shown in Table III for file size smaller than 10 kB. The case of files larger than 10 kB is omitted. Columns in the Table show: 1) emulated drop probability; 2) emulated latency; 3) samples number (number of flows); 4) estimate of the average Retransmission Timer (RT) at the end of the connection; 5) percentage of times a loss resulted in an RTO expiration; and 6) Completion Time difference $\Delta\mathrm{CT}_{\mathrm{loss}}$ between the two examined TCP versions, computed only for flows that have experienced at least one loss event.

Observe that positive values for the Completion Time difference indicate that TCP-SF exhibited a faster Completion Time, while negative times indicate the opposite.

Overall, results confirm the findings shown in previous sections via simulation and, specifically, identify TCP-SF as less prone to RTO expirations than Classic TCP. Estimating the proper value of the Retransmission Timer also benefits from the features of TCP-SF: the larger number of segments sent, compared to Classic TCP, accounts for a larger number of samples used in the estimation of the round trip time, thus refining the estimate and providing a smaller, more accurate value for the timer. The combined effects of fewer RTOs and smaller values of the retransmission timer shortens the Completion Time in presence of a loss; although we remarked that Completion Times are affected by the unpredictable behavior of different browser types, the trend nonetheless confirms a sizable reduction of Completion Times for TCP-SF.

## V. ANALYSIS OF THE SMALL WINDOW REGION

In this section we analytically evaluate the behavior of TCP-SF and of other classic versions of TCP in the small window regime. The analysis is based on the computation of the throughput achieved by the protocol given the network conditions; i.e., given a value of the average round-trip time and of the segment loss probability. The small window region is defined by window sizes ranging between 1 MSS and $\mathrm{cwnd}_{\min}$ (for ease of computation, from now on all window sizes are expressed in multiples of MSS). Out of this region, all TCP versions exhibit the same behavior, and further gains are unlikely to be achieved by TCP-SF, unless combinations of losses of both segments and ACKs make the use of TCP-SF still profitable. For simplicity, we assume that the congestion window grows always according to the Slow Start algorithm, since in the small window region the exponential and linear growth of cwnd are indeed very similar.

In order to describe the behavior of Classic TCP, of FS-TCP-SF and of VS-TCP-SF for small values of the window size, we develop three models based on the state machine of TCP congestion control, following a procedure similar to the one proposed in [21] and [22]. The idea is to evaluate the time spent and the data sent by the different versions of TCP in the small window region only. A continuously backlogged source is assumed and no ACKs are lost.

### A. Model of Classic TCP

We now present the State Machine (SM) which describes the behavior of classic versions of TCP when cwnd is comprised be-

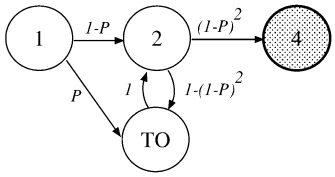



Fig. 9. State machine of Classic TCP in the small window region.

tween 1 and the threshold $\mathrm{cwnd}_{\min} = 4$. When developing the key ideas of the SM, we focus on the state changes neglecting the time spent in each state, which will be introduced later.

The state machine of Classic TCP is shown in Fig. 9. At the connection set up (as well as when Slow Start is entered), $\mathrm{cwnd} = 1$; in the SM this is represented by state 1. One segment is transmitted and, after a round-trip time, if the segment was not lost, TCP increases cwnd to 2. Correspondingly, the SM moves from state 1 to state 2 with probability $(1 - P)$, where $P$ is the segment loss probability. Similarly, the transition from state 2 to state 4 has probability $(1-P)^2$. State TO represents the timeout. If the segment retransmitted after timeout expiration is lost again, the SM remains in TO and the backoff mechanism applies. When the retransmitted segment is successfully delivered, the SM leaves state TO and enters state 2, because the new ACK makes cwnd slide and widen.

We now evaluate the average time $D_c$ needed by the protocol to move from state 1 to state 4, since, from that moment on, Classic TCP and TCP-SF behave the same. From the SM, we observe all possible paths between 1 and 4 and we evaluate their probability. We then weight each path by its duration, i.e., by the sum of the times needed to visit all the states along the path.

The duration of a visit to states 1 and 2 is equal to the round-trip time RTT, $r$ for short in the following formula; a visit to state TO, instead, takes a time equal to the average timeout duration $T$. Taking into account the backoff mechanism, and stating by RTO the value of the Retransmission timer estimation

$$T = \mathrm{RTO}\sum_{i=1}^{\infty} 2^{i-1}P^{i-1}(1-P) = \frac{\mathrm{RTO}(1-P)}{1-2P}. \tag{1}$$

We obtain

$$\begin{aligned} D_c &= (1-P)r + P(T+r) + (1-P)^2 \\ &\quad \times \sum_{i=0}^{\infty}[1-(1-P)^2]^i[i(T+r)+r] \\ &= (1-P)r + P(T+r) + \frac{(T+r)[1-(1-P)^2]}{(1-P)^2} + r. \end{aligned}$$

The infinite sum accounts for losses occurring in state 2.

In a similar way, it is possible to compute $N_c$, the average number of segments successfully transmitted in the small window region. We again consider all possible paths from state 1 to state 4 and for each visited state along a path we compute the number of transmitted segments

$$\begin{aligned} N_c &= P + (1-P) + (1-P)^2\sum_{i=0}^{\infty}[1-(1-P)^2]^i \\ &\quad \times [i(\bar{N}+1)+2] \qquad (2) \\ &= 3 + \frac{(\bar{N}+1)[1-(1-P)^2]}{(1-P)^2} \qquad (3) \end{aligned}$$

where $\bar{N} = 2P(1-P)/[1-(1-P)^2]$ is the average number of segments successfully transmitted in state 2 given that some losses occur. In (2), the term $P+(1-P)$ represents the average number of segments transmitted when the chain moves from 1 to 2 (either directly or passing through state TO); term $i$ in the infinite sum accounts for the segments transmitted when the timeout expiration is required $i$ times before the protocol manages to reach state 4 (thus exiting from the small window regime). We can now compute the throughput in the small window region as the ratio between $N_c$ and $D_c$

$$X_c = \mathrm{MSS}\frac{N_c}{D_c}. \tag{4}$$

The expressions for $N_c$ and $D_c$ can be obtained through an alternative formulation of the problem, which is based on the observation that the behavior of the SM forms a semi-Markov chain. In order to derive the embedded Markov chain, we define the following ordering of the states: $\langle 1, 2, \mathrm{TO}, 4\rangle >$ and derive, accordingly, the transition probability matrix $\mathbf{P}_c$. The probabilities $\pi_s^{(i)}$ that the $i$th state visited by the protocol is $s$ are collected in the vector $\boldsymbol{\pi}_c^{(i)} = [\pi_1^{(i)}\ \pi_2^{(i)}\ \pi_{\mathrm{TO}}^{(i)}\ \pi_4^{(i)}]$. Since the small window regime is started in state 1, the initial probability vector is $\boldsymbol{\pi}_c^{(0)} = [1\ 0\ 0\ 0]$.

We now associate a time reward to each state which coincides with the time spent by the protocol in that state for each visit. The column vector of the time reward is $\mathbf{d}_c = [r\ r\ T\ 0]^{\mathrm{T}}$. We can derive $D_c$ as

$$\begin{aligned} D_c &= \sum_{i=0}^{\infty}\boldsymbol{\pi}_c^{(i)}\mathbf{d}_c = \boldsymbol{\pi}_c^{(0)}\sum_{i=0}^{\infty}\mathbf{P}_c^{(i)}\mathbf{d}_c \qquad (5) \\ &= \boldsymbol{\pi}_c^{(0)}(\mathbf{I}-\mathbf{P}_c)^{-1}\mathbf{d}_c \qquad (6) \end{aligned}$$

where $\mathbf{I}$ is the identity matrix.

Similarly, in order to derive $N_c$ we introduce the data reward vector $\mathbf{n}_c$ which collects the number of segments successfully transmitted in each state. We derive

$$\mathbf{n}_c = [P \quad (2P(1-P)+2(1-P)^2) \quad 1 \quad 0]^{\mathrm{T}}. \tag{7}$$

$N_c$ can now be computed as, $N_c = \boldsymbol{\pi}_c^{(0)}(\mathbf{I}-\mathbf{P}_c)^{-1}\mathbf{n}_c$. Given $D_c$ and $N_c$, the throughput is given as before by (4).

Another interesting performance metric is the *waste of bandwidth*, which is the amount of information submitted to the network but not delivered to the end user. It is given by the sum of two terms: the bandwidth used by TCP and IP overheads in each segment ($\mathrm{OH} = 40$ bytes), which is given by $\mathrm{OH}(N_c/D_c)$, and the bandwidth wasted for the transmission of lost segments. In order to evaluate this second contribution, we compute the offered traffic from which we subtract the throughput. Thus, we introduce the reward vector $\mathbf{t}_c$ which defines the number of segments generated in each state of the Markov chain, $\mathbf{t}_c = [1\ 2\ 1\ 0]^{\mathrm{T}}$. We now compute the offered traffic $T_c$ as

$$T_c = \boldsymbol{\pi}_c^{(0)}\sum_{i=0}^{\infty}\mathbf{P}_c^{(i)}\mathbf{t}_c = \boldsymbol{\pi}_c^{(0)}(\mathbf{I}-\mathbf{P}_c)^{-1}\mathbf{t}_c. \tag{8}$$

The waste of bandwidth results

$$\begin{aligned} O_c &= \mathrm{OH}\frac{N_c}{D_c} + \left(\mathrm{MSS}\frac{T_c}{D_c} - X_c\right) \\ &= \frac{\mathrm{MSS}\cdot T_c - (\mathrm{MSS}-\mathrm{OH})\cdot N_c}{D_c}. \end{aligned} \tag{9}$$

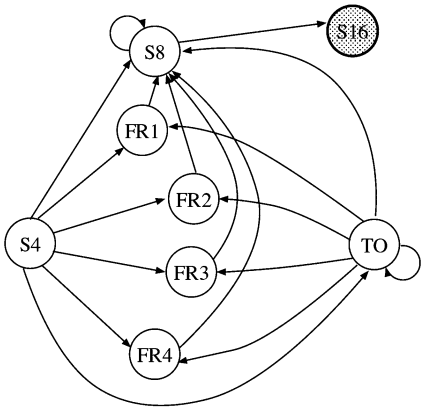



Fig. 10. State machine of FS-TCP-SF in the small window region.

### *B. Model of FS-TCP Smart Framing*

The state machine diagram which describes FS-TCP-SF behavior is shown in Fig. 10.

When entering Slow Start cwnd is equal to $\mathrm{cwnd}_0$ and TCP-SF transmits four small segments. In the SM, we represent this situation by state $S4$.

When losses occur, TCP-SF can either enter FR or RTO; this choice depends on the loss pattern, and in particular it changes on the basis of which segment within the window is the first one to be lost. In order to distinguish different cases, we introduce in the SM four states labeled as $\mathrm{FR}_i$ with $i = 1, \ldots, 4$, where $\mathrm{FR}_i$ represents the FR state entered by TCP when the first lost segment is the $i$th one.

State $\mathrm{FR}_1$ represents the case in which the first segment out of four is lost. In this case, FR can be entered only if the following three segments are successfully delivered. Therefore, the transition probability from state $S4$ to $\mathrm{FR}_1$ is equal to $p(S4, \mathrm{FR}_1) = P_s(1 - P_s)^3$, where $P_s$ is the small-size segment loss probability. In case the second segment of the window is lost, the ACK of the first segment makes the window slide and widen so that a total number of four segments have been transmitted after the lost one. Therefore, FR is entered if at least three out of the four segments transmitted after the lost one are successfully delivered so that a total number of three duplicated ACKs are generated. In the SM, the state $\mathrm{FR}_2$ represents this case. Similar situations are represented by states $\mathrm{FR}_3$ and $\mathrm{FR}_4$.

We denote by $p(S4, \mathrm{FR}_i)$ the transition probability from state $S4$ to $\mathrm{FR}_i$; we have

$$p(S4, \mathrm{FR}_i) = (1 - P_s)^{i-1} P_s \sum_{j=0}^{i-1} \binom{i+2}{j} (1 - P_s)^{i+2-j} P_s^j.$$

Since in state $\mathrm{FR}_i$ the number of segments which follow the lost one is $i + 2$, the sum accounts for the probability that at least 3 segments following the lost one are correctly received, so that FR can be triggered.

Once the FR is completed, the SM will always move to the state labeled by $S8$, where 8 small segments are sent. This is due to the clipping of the Slow Start threshold to 2MSS.

The transition from state $S4$ to state $S8$ represents the case in which all four segments transmitted when the window size is equal to 4 are successfully delivered so that, after roughly a round-trip time, the window size is equal to 8 small segments. The transition probability is equal to $p(S4, S8) = (1 - P_s)^4$.

If a loss pattern occurs so that FR cannot be entered, the SM moves from state $S4$ to state TO; the probability of this transition is

$$p(S4, \mathrm{TO}) = 1 - \sum_{i=1}^{4} p_{S4,\mathrm{FR}i} - (1 - P_s)^4.$$

The SM can then leave state TO and move to state $S8$ if no loss occurs, or, similarly to what happens from state $S4$, the SM can move to a state $\mathrm{FR}_i$, if three duplicate ACKs arrive, or remain in the state TO otherwise.

When the window size is 8, the probability that RTO is used instead of FR is extremely small, therefore, we assume that in case of losses FR only is possible. Loss and recovery, when the window size is equal to 8, are represented by the transition from state $S8$ to itself; the transition probability is approximated by $p(S8, S8) = 1 - (1 - P_s)^8$. Notice that the window size does not change due to the minimum Slow Start threshold, which is equal to 2MSS. By ordering the states as

$$\langle S4, \mathrm{FR}_1, \mathrm{FR}_2, \mathrm{FR}_3, \mathrm{FR}_4, \mathrm{TO}, S8, S16 \rangle$$

we can write the transition probability matrix of the embedded Markov chain, $\mathbf{P}_f$ and we can define the state probability vectors $\boldsymbol{\pi}_f^{(i)}$. The initial probability vector is $\boldsymbol{\pi}_f^{(0)} = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$.

Similarly to (4) in the previous model, the throughput is evaluated based on the average number of transmitted segments, $N_f$, and the average time spent by TCP-SF in the considered region, $D_f$. The protocol spends a round trip time $r$ for each visit to states $S4$, $S8$, and $\mathrm{FR}_1$, and two round trip times for each visit to states $\mathrm{FR}_i$, with $i > 1$ (one round trip time is needed to receive the ACKs which enlarge the window and allow the transmission of new segments; one more round trip time is needed to receive the ACKs of these new segments). The protocol spends a timeout in TO. Thus, the vector of the time rewards is given by, $\mathbf{d}_f = [r\ r\ 2r\ 2r\ 2r\ T\ r\ 0]^{\mathrm{T}}$. We compute $D_f$ as

$$D_f = \boldsymbol{\pi}_f^{(0)} \sum_{i=0}^{\infty} \mathbf{P}_f^{(i)} \mathbf{d}_f = \boldsymbol{\pi}_f^{(0)} (\mathbf{I} - \mathbf{P}_f)^{-1} \mathbf{d}_f. \quad (10)$$

For the derivation of the data reward vector, we compute the number of segments successfully transmitted in each state. In states $S4$ and TO, the number of transmitted segments, respectively, $\mathbf{n}_f(S4)$ and $\mathbf{n}_f(\mathrm{TO})$, is

$$\mathbf{n}_f(S4) = \mathbf{n}_f(\mathrm{TO}) = \sum_{i=1}^{4} i \binom{4}{i} (1 - P_s)^i P_s^{4-i}.$$

In states $\mathrm{FR}_i$, one segment is transmitted with probability $1 - P_s$; thus, $\mathbf{n}_f(\mathrm{FR}_i) = 1 - P_s$. Finally, for state $S8$ we have

$$\mathbf{n}_f(S8) = \sum_{i=1}^{8} i \binom{8}{i} (1 - P_s)^i P_s^{8-i}.$$

Thus, the average number of successfully transmitted segments is, $N_f = \boldsymbol{\pi}_f^{(0)} (\mathbf{I} - \mathbf{P}_f)^{-1} \mathbf{n}_f$. We compute the throughput in the small window region as

$$X_f = \frac{\mathrm{MSS}}{4} \frac{N_f}{D_f}. \quad (11)$$

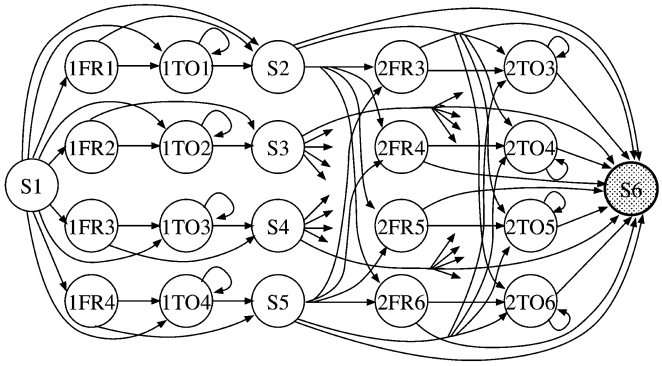



Fig. 11. State machine of VS-TCP-SF in the small window region.

In order to evaluate the waste of bandwidth due to overhead, we first derive the offered traffic. Then, by subtracting the throughput to the offered traffic we compute the waste of bandwidth due to lost segments, from which we get the total waste of bandwidth by simply adding the amount of bandwidth used for protocol headers. We introduce the reward vector $\mathbf{t}_f$ which defines the number of segments generated in each state of the Markov chain, $\mathbf{t}_f = [4\ 1\ 1\ 1\ 1\ 4\ 8\ 0]^{\mathrm{T}}$.

We compute the offered traffic $T_f$ as

$$T_f = \boldsymbol{\pi}_f^{(0)} \sum_{i=0}^{\infty} \mathbf{P}_f^{(i)} \mathbf{t}_f = \boldsymbol{\pi}_f^{(0)} (\mathbf{I} - \mathbf{P}_f)^{-1} \mathbf{t}_f \qquad (12)$$

and the waste of bandwidth as

$$O_f = \mathrm{OH} \frac{N_f}{D_f} + \left( \frac{\mathrm{MSS}}{4} \frac{T_f}{D_f} - X_f \right) = \frac{\mathrm{MSS}/4 \cdot T_f - (\mathrm{MSS}/4 - \mathrm{OH}) \cdot N_f}{D_f}. \qquad (13)$$

### C. Model of VS-TCP Smart Framing

In order to model the behavior of VS-TCP-SF we have to account for the different sizes that the segments may have. By assuming, as in previous models, $\mathrm{cwnd}_{\min} = 4$, the segment size can have six different values $M_i = (\mathrm{MSS}/4) * \alpha^{i-1}, i = 1, \ldots, 5$ with $\alpha = 1.32$, and $M_i = \mathrm{MSS}, \forall i \geq 6$. In general, segments with different size may experience different loss probability; hence, we denote by $P_i$ the loss probability of segments whose size is equal to $M_i$, with $i = 1, \ldots, 6$. The case of $i = 1$ corresponds to the small segment size employed by FS-TCP-SF i.e., $M_1 = \mathrm{MSS}/4$ and $P_1 = P_s$; the case $i = 6$ corresponds to the full-size segment, $M_6 = \mathrm{MSS}$ and $P_6 = P$.

The Markov chain describing the behavior of VS-TCP-SF in the small window region is shown in Fig. 11. State $S1$ in the Markov chain represents the transmission of 4 small segments when Slow Start is entered. States $1\mathrm{FR}i$ and $1\,\mathrm{TO}\,i$ refer to the transmission of the first window, and stand for the cases in which the $i-$th segment is the first one to be lost and either a Fast Recovery or a Timeout phase is entered. When the loss is recovered from states $1\,\mathrm{FR}i$ or $1\,\mathrm{TO}\,i$, the chain moves to states $Sj$ with $j = 2, \ldots, 5$ which correspond to a new window whose first segment size is equal to $M_j$. For example, suppose that the second segment is lost while the first one as well as the segments following the second one are correctly received. The ACK of the first segment makes the window slide and widen so that a segment with size $M_2$ is transmitted and FR is entered (state 1FR2). When the lost segment has been successfully transmitted, the new window starts from a segment whose size is $M_3$ (state $S3$). Therefore, the Markov chain visits the states $S1$, 1FR2, and $S3$. Similar behaviors are associated to states $2\,\mathrm{FR}\,i$ and $2\,\mathrm{TO}\,i$ with $i = 3, \ldots, 6$ which represent the FR and timeout procedure to recover the loss of a segment with size $M_i$ in the second transmitted window. The small window regime terminates in state $S6$, in which the first segment of the new window is full-size.

For the sake of brevity, we do not report here the expressions for the transition probabilities between states, the small window regime throughput and the waste of bandwidth, which are derived by considering all possible loss patterns as we did for FS-TCP-SF.

### D. Comparisons

Performance is going to be assessed in terms of throughput and waste of bandwidth due to overhead. Since segment loss probability has a major impact on the performance and it may depend on the segment size, we need to know the relationship between the segment loss probability and the segment size. If RED buffer management schemes are adopted and the queue length is managed in bytes, it is reasonable to assume that the segment loss probability is proportional to the segment size, as suggested in [23]. Then, for example, the probability to lose a small segment, $P_s$, is about 4 times smaller than the probability of losing a full-size segment $P$. On the contrary, for a buffer management scheme which handles packets as units, independently of their size, $P_s$ is equal to $P$. Some other factors may induce different relationships between segment loss probability and segment size. However, we will consider only the two cases mentioned above. In the figures, the label "linear" will indicate the case of loss probability proportional to the segment size (this is the most favorable case for TCP-SF), the label "constant" will indicate the case in which the segment loss probability is independent from the segment size.

The first performance metric which we consider for the comparison is the throughput gain that TCP-SF can achieve over classic versions of TCP:

$$G_f = 100 \frac{X_f - X_c}{X_c} \quad \text{and} \quad G_v = 100 \frac{X_v - X_c}{X_c}$$

where $G_f$ and $G_v$ are the throughput gain of FS-TCP-SF and VS-TCP-SF, respectively.

Fig. 12 shows the throughput gain of TCP-SF over Classic TCP in the small window regime versus the full-size segment loss probability for the case of loss probability proportional to the segment size or constant. The average round trip time is equal to 200 ms, RTO is set to 1.5 s. The chosen values of segment loss probability were taken from simulation results for different values of the offered load; see Section IV. As expected, TCP-SF benefits from the loss probability proportionally to the segment size, since small packets experience a larger chance to be successfully delivered. However, positive gains can be observed even if the loss probability is constant. When the segment loss probability is large, the FS option of TCP-SF provides a sizable gain over the VS one, since recovering losses by Fast Recovery is more likely than by RTO expiration. The negative gain which can be observed when the segment loss probability is very small, indicates that the throughput of TCP-SF is smaller than that of Classic TCP. This is due to the effect of the larger

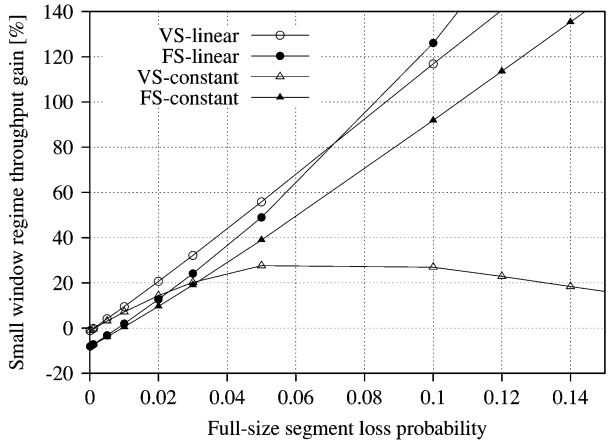



Fig. 12. Throughput gain in the small window region for FS-TCP-SF and VS-TCP-SF over Classic TCP.

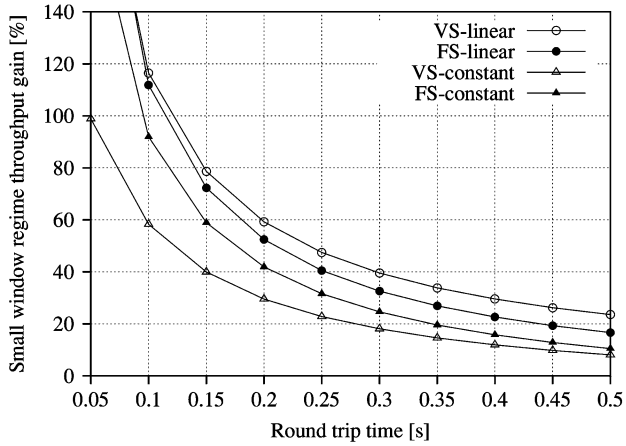



Fig. 13. Throughput gain in the small window region for FS-TCP-SF and VS-TCP-SF over Classic TCP.

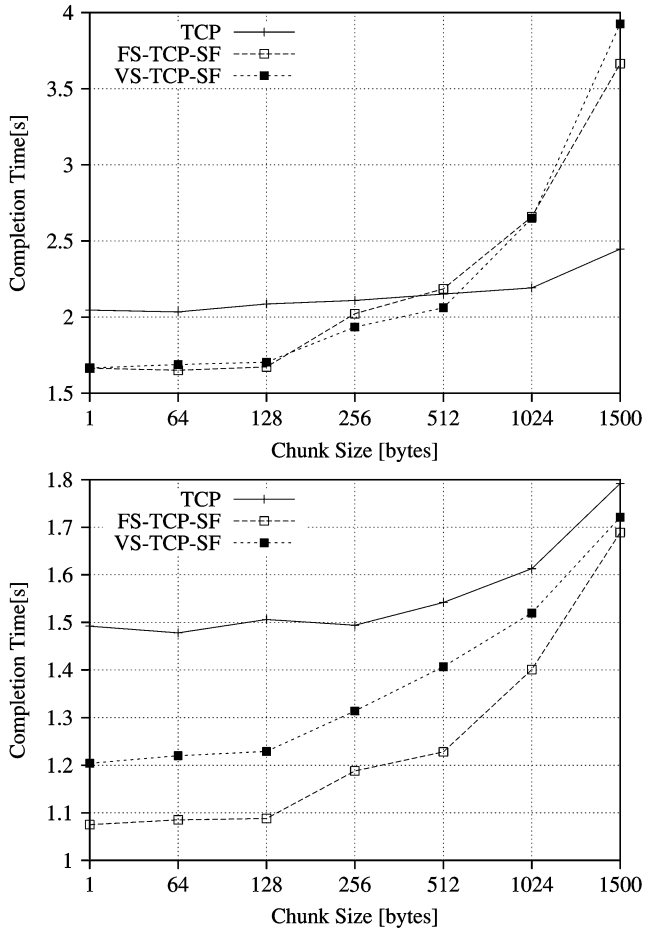



Fig. 14. Average Completion Time versus chunk size: (top) worst case scenario and (bottom) shaped scenario.

overhead introduced by TCP-SF. Notice, however, that the cases with small loss probabilities are less critical from the point of view of a service provider, since the QoS offered to the end user is already high.

Since the advantage of TCP-SF is mainly due to the smaller time spent in FR rather than waiting for RTO expiration, we now focus on the impact of the values of RTT and RTO on the throughput gain. In Fig. 13, the throughput gain is plotted versus the RTT for RTO equal to 1.5 s and full-size segment loss probability $P$ equal to 0.05. As the round trip time increases, the difference between RTO and RTT decreases; consistently, the throughput gain decreases. Notice however, that for the considered scenario, the throughput gain is still remarkable when the RTT is 0.5 s, i.e., only one third of the RTO.

Finally, note that short-lived flows enjoy further advantages by using TCP-SF that are not considered in this analysis, such as the higher chance not to suffer the initial RTO. For longer flows, on the contrary, the benefits of adopting the smart framing option are observed not only at connection set up but also when Slow Start is entered due to some losses. Thus, the advantage perceived by the end user (on the Completion Time of the connection) is smaller.

## VI. Router Memory Management Issues

In this section, we address some of the issues related to how routers store and handle incoming packets, and to the impact such strategies have on TCP-SF performance. Since router buffers can be organized at the granularity of "chunks" larger than a single byte, the increased packet count due to smart framing may affect buffer occupancy, reducing the benefits of TCP-SF. Since reasons of space do not allow us to explore all possible buffer management policies in the present work, we investigate the impact of memory management in Drop-Tail buffers only. We therefore conduct additional simulations, in which router buffers are managed considering chunks of $\{1, 64, 128, 256, 512, 1024, 1500\}$ bytes, i.e., from a pure byte-oriented to a pure packet-oriented fashion.

We consider the same bottleneck link topology seen in previous sections, in which the Drop-Tail buffer associated to the congested link has a total capacity of 150 000 bytes. For each incoming packet, an amount of memory, corresponding to an integer number of chunks, is allocated from the buffer space. For example, when 256-byte-long chunks are used, a 1500-byte-long packet requires six chunks to be accommodated in the buffer, while two chunks are required for a 375-byte-long packet. The load offered by TCP sources (directly connected to the nodes at bottleneck link) is fixed and equal to 80% of bottleneck capacity.

Results are plotted at the top of Fig. 14, that reports the absolute average completion time versus chunk size. It can be noticed that the Smart Framing versions of TCP improve the performance only when the chunk size is smaller than 512 bytes,

while the completion time increases up to about 175%, when the buffer is managed in packets, i.e., the chunk size is equal to 1500 bytes and the bottleneck queue stores up to 100 packets. This degradation in performance is due to two reasons. First, the generation process of the packets that Smart Framing sources send in the network is burstier, since they are allowed to send four 375-byte-long segments; this translates into the request to allocate four buffer chunks of 1500 bytes all at once. Second, by allocating 1500 bytes per 375-byte-long segment, a worse usage of the router memory is achieved, since the effective memory usage is only $1/4$ that of classic TCP.

For what concerns the first reason mentioned above, the impact of the higher burstiness of the packet generation process of Smart Framing is amplified by the considered scenario which is a sort of unrealistic, worst-case scenario. In it TCP sources inject back-to-back packets directly in the bottleneck queue, without any traffic mixing and multiplexing by previous nodes. We therefore consider a more realistic scenario, which differs from the previous one only because TCP sources, instead of being connected to the same access node and then to the bottleneck link, are now connected to one of 10 possible access nodes; the access nodes are then connected to the bottleneck by links with 1.5 Mb/s capacity and 1 ms propagation delay. Results are reported in the bottom plot of Fig. 14, where TCP-SF outperforms Classic TCP over all chunk sizes, even the largest one. Notice also that all TCP variants benefit of the reduced burstiness at the packet level achieved by assuming the more realistic scenario with 10 access nodes. These results confirm the positive findings from live measurements discussed in Section IV.

## VII. Impact of Protocol Overhead

We now explicitly consider the waste of bandwidth due to the protocol overhead, which is the major drawback of TCP-SF. For example, when no losses occur and for flows that can exit the small-window-regime, FS-TCP-SF sends 28 small-size segments before switching back to large-size segments, VS-TCP-SF sends 12 segments, while Classic TCP always sends 7 segments. This translates into a 300% protocol overhead increase for FS-TCP-SF with respect to Classic TCP and 70% for VS-TCP-SF. However, this is a worst case analysis. By considering: 1) the real traffic distribution obtained in Section II; 2) a segmentation based on $\mathrm{MSS} = 1460$ bytes; 3) IP/TCP protocol overhead of 40 bytes; and 4) by assuming that no segment is lost, the percentage of the average waste of bandwidth due to protocol overhead accounts to: 5.6% for Classic TCP, 8.0% for FS-TCP-SF, and 6.5% for VS-TCP-SF. Thus, if we compare the bandwidth waste of Classic TCP and VS-TCP-SF, the latter scores performance improvements up to 35% at the cost of 3% increase of bandwidth waste. In order to also consider the impact of loss probability on the waste of bandwidth, we employ the analytical model proposed in the previous section, which, we recall, refers to the small window region only.

In Fig. 15, we plot the percentage of the bandwidth employed for the overhead versus the full-size segment loss probability. The value corresponding to loss probability equal to 0 represents the amount of bandwidth needed to carry the additional bytes in the protocol headers. As the loss probability increases, the waste of bandwidth accounts also for the bytes retransmitted due to lost segments. Clearly, the waste of bandwidth of TCP-SF is larger when the loss probability is constant instead of linear with the segment size, since in this case the loss probability results to be larger for TCP-SF. The largest waste of bandwidth can be observed for FS-TCP-SF; for example, when the loss probability is small, FS-TCP-SF wastes 10% of the available bandwidth in overhead in the small window regime region, which is 3 times what is wasted by Classic TCP, and twice the waste of VS-TCP-SF. Notice, however, that, in the linear case with loss probability larger than 0.08, the waste of Classic TCP is larger than that of VS-TCP-SF; when the loss probability reaches 0.12, the waste of Classic TCP is even larger than that of FS-TCP-SF.

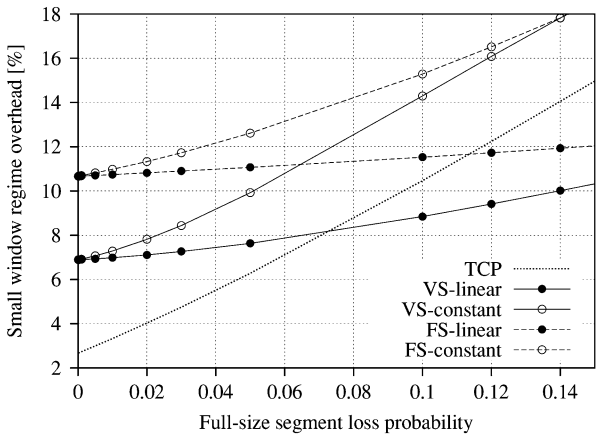



Fig. 15. Bandwidth wasted for the protocol overhead in the small window region for TCP-SF and Classic TCP.

In summary, we have shown that, under several common scenarios and settings, the improvements achieved by the smart segmentation algorithm of TCP-SF are worth the cost of the overhead increase. In addition, the gain appears to be higher when the network is congested and translates into a better exploitation of resources.

## VIII. Conclusion

We proposed an enhancement to the TCP protocol that is based on the key idea of transmitting small-size segments when the congestion window is small, without changing the degree of aggressiveness of the source. This allows the sender to receive more feedback from the destination, and thus use the Fast Recovery algorithm to recover from segment losses, without waiting for a Retransmission Timeout expiration to occur.

TCP-SF is particularly effective for short-lived flows, but improves the responsiveness of long file transfers also. In a current Internet traffic scenario, TCP-SF outperforms Classic TCP in terms of both Completion Time, and probability to trigger Fast Recovery to detect segment losses, stealing little or no bandwidth from Classic TCP. In highly congested networks, it also guarantees a better exploitation of resources.

The proposed modification is extremely simple and can be implemented on top of any TCP flavor, and requires changes on the server side only.

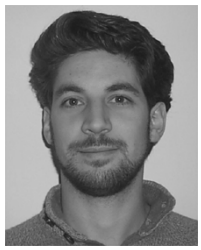
## Acknowledgment

Hands-on measurements of Internet traffic could not have been possible without the cooperation of several people from

our department who willingly (or unaware) left their browsers at our mercy for a month or so. In acknowledging their help, we also wish to thank the Politecnico di Torino Network Facilities that allowed us to take the network measurements, and Mario Gerla at UCLA and the Computer Science Facility at CMU for letting us run remote experiments on their workstations. Finally, we acknowledge the valuable comments and suggestions of the anonymous reviewers.

## REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," IETF, RFC 2581, Apr. 1999.
[2] S. Floyd, "A report on recent developments in TCP congestion control," *IEEE Commun. Mag.*, vol. 39, no. 4, pp. 84–90, Apr. 2001.
[3] V. Paxson and M. Allman, "Computing TCP's retransmission timer," IETF, RFC 2988, Nov. 2000.
[4] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, Stanford, CA, Aug. 1988, pp. 314–329.
[5] W. R. Stevens, *TCP/IP Illustrated*. Reading, MA: Addison Wesley, 1994, vol. 1.
[6] S. Floyd and T. Henderson, "The NEWRENO modification to TCP's fast recovery algorithm," IETF, RFC 2582, Apr. 1999.
[7] M. Mathis, J. Mahdavi, S. Floyd, and S. A. Romanow, "TCP selective acknowledgment options," IETF, RFC 2018, Apr. 1996.
[8] M. Mellia, A. Carpani, and R. Lo Cigno, "Measuring IP and TCP behavior on edge nodes," in *Proc. IEEE GLOBECOM*, Taipei, Taiwan, Nov. 2002.
[9] Tstat Home Page [Online]. Available: http://tstat.tlc.polito.it
[10] GARR—The Italian Academic and Research Network [Online]. Available: http://www.garr.it
[11] A. Feldmann, J. Rexford, and R. Caceres, "Efficient policies for carrying web traffic over flow-switched networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 6, pp. 673–685, Dec. 1998.
[12] W. Willinger, V. Paxson, and M. S. Taqqu, "Self-similarity and heavy tails: Structural modeling of network traffic," in *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, R. J. Adler, R. E. Feldman, and M. S. Taqqu, Eds. Basel, Switzerland: Birkhauser, 1998.
[13] Cooperative Association for Internet Data Analysis [Online]. Available: http://www.caida.org
[14] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's loss recovery using limited transmit," IETF, RFC 3042, Jan. 2001.
[15] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's initial window," IETF, RFC 3390, Oct. 2002.
[16] K. Poduri and K. Nichols, "Simulation studies of increased initial TCP window size," IETF, RFC 2415, Sep. 1998.
[17] M. Allman, K. Avrachenkov, U. Ayesta, and J. Blanton, "Early retransmit for TCP and SCTP," IETF, Draft, work in progress, Aug. 2003.
[18] M. Allman, "TCP byte counting refinements," *ACM Comput. Commun. Rev.*, vol. 29, no. 3, Jul. 1999.
[19] ns-2, network simulator (ver.2) [Online]. Available: http://www.isi.edu/nsnam/ns
[20] M. Mellia, C. Casetti, and M. Meo, "TCP smart framing: Using smart segments to enhance the performance of TCP," in *Proc. IEEE GLOBECOM*, San Antonio, TX, Nov. 2001.
[21] M. Mellia, I. Stoica, and H. Zhang, "TCP model for short lived flows," *IEEE Commun. Lett.*, vol. 6, no. 2, pp. 85–88, Feb. 2002.
[22] P. Barford and M. Crovella, "Critical path analysis of TCP transactions," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 238–248, Jun. 2001.
[23] S. De Cnodder, O. Elloumi, and K. Pauwels, "RED behavior with different packet sizes," in *Proc. 5th IEEE Symp. Computers and Communications (ISCC)*, Jul. 2000.



**Marco Mellia** (M'97) received the Ph.D. degree in electronic and telecommunication engineering in 2001 from the Politecnico di Torino, Italy.

From March to October 1999, he was with the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, as a Visiting Scholar. Since April 2001, he has been with the Electronics Department of the Politecnico di Torino as an Assistant Professor. His research interests are in the fields of all-optical networks, traffic measurement and modeling, and QoS routing algorithms.

**Michela Meo** (M'98) received the Ph.D. degree in electronic and telecommunication engineering in 1997 from Politecnico di Torino, Italy.

Since then she has been working in the Telecommunication Networks Research Group of the Politecnico di Torino, where she is currently an Assistant Professor. Her research interests are in the field of performance evaluation of communication networks with a particular focus on wireless systems and end-to-end performance.

**Claudio Casetti** received the Ph.D. degree in electronic and telecommunication engineering from the Politecnico di Torino, Italy, in 1997.

He is an Assistant Professor at the Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino. His interests are in the field of performance evaluation of TCP/IP networks and wireless communications.