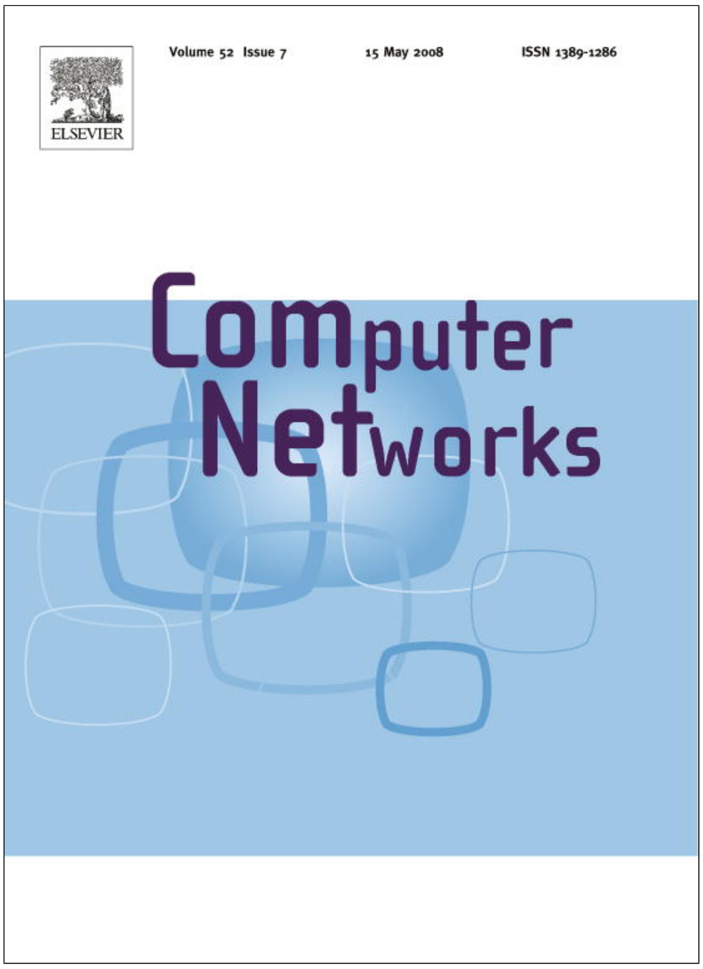

Volume 52 Issue 7
15 May 2008
ISSN 1389-1286
ELSEVIER
Computer Networks

Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 52 (2008) 1473–1491

Computer Networks

www.elsevier.com/locate/comnet

Loss-resilient window-based congestion control [☆]

Christophe De Vleeschouwer, Pascal Frossard [*]

Université Catholique de Louvain, UCL – TELE, Louvain-la-Neuve 1348, Belgium
Ecole Polytechnique Fédérale de Lausanne (EPFL), Signal Processing Laboratory – LTS4, Lausanne 1015, Switzerland

Received 14 March 2007; received in revised form 30 November 2007; accepted 21 January 2008
Available online 5 February 2008

Responsible Editor: Qian Zhang

Abstract

This paper addresses the problem of fair allocation of bandwidth resources on lossy channels in hybrid heterogeneous networks. It discusses more particularly the ability of window-based congestion control to support non-congestion related losses. We investigate methods for efficient packet loss recovery by retransmission, and build on explicit congestion control mechanisms to decouple the packet loss detection from the congestion feedback signals. For different retransmission strategies that respectively rely on conventional cumulative acknowledgments or accurate loss monitoring, we show how the principles underlying the TCP retransmission mechanisms have to be adapted in order to take advantage of an explicit congestion feedback. A novel retransmission timer is proposed in order to deal with multiple losses of data segments and, in consequence, to allow for aggressive reset of the connection recovery timer. It ensures significant benefit from temporary inflation of the send-out window, and hence the fair share of bottleneck bandwidth between loss-prone and lossy connections. Extensive simulations analyze the performance of the new loss monitoring and recovery strategies, when used with two distinct explicit congestion control mechanisms. The first one relies on a coarse binary congestion notification from the routers. The second one, introduced in [D. Katabi, M. Handley, C. Rohrs, Internet congestion control for high bandwidth-delay product environments, ACM SIGCOMM (2002) 89–102], exploits accurate and finely-tuned router feedbacks to compute a precise congestion window adjustment. For both congestion control mechanisms, we observe that retransmissions triggered based on a precise monitoring of losses lead to efficient utilization of lossy links, and provide a fair share of the bottleneck bandwidth between heterogeneous connections, even for high loss ratios and bursty loss processes. Explicit window-based congestion control, combined with appropriate error control strategies, can therefore provide a valid solution to reliable and controlled connections over lossy network infrastructures.
© 2008 Elsevier B.V. All rights reserved.

Keywords: Congestion control; Lossy networks; Window-based protocol; Explicit notification

1. Introduction

Congestion control is certainly imperative in packet networks, as it prevents important bandwidth outage that happens when the network is overwhelmed by too many packets. In addition, it

[☆] This work has been funded by the Swiss NSF, under Grant PP002-68737, and by the Belgian NSF.

[*] Corresponding author. Address: Ecole Polytechnique Fédérale de Lausanne (EPFL), Signal Processing Laboratory – LTS4, Lausanne 1015, Switzerland.
E-mail address: pascal.frossard@epfl.ch (P. Frossard).

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.01.013

tends to fairly distribute bandwidth resource among simultaneous connections and users, and avoids any one connection from swamping the links and switches between communicating hosts with an excessive amount of traffic. Two families of approaches have been proposed to implement congestion control. Rate-based algorithms directly control the transmission rate of the connection [2,3], whilst window-based algorithms force the connection to obey a 'packet conservation' principle, which means that a new packet is not pushed into the network until an old packet leaves [4]. Typically, window control injects bursts of data into the network, whilst rate control pushes data at regular time interval so as to meet a target average rate. More importantly, window-based protocols are known to achieve network stability and present a reduced sensitivity (or increased robustness) to inaccurate bandwidth estimation, in comparison to rate-based congestion control schemes. If the sources implement a rate-based congestion control algorithm but over-estimate the available bandwidth, the network could temporarily experience an extremely high packet loss rate due to buffer overflows and may take a long time to recover from it. In contrast, a window-based congestion control algorithm not only controls the transmission rate, but also limits the maximum number of outstanding packets according to the congestion window size. It alleviates the long term effect of the inaccurate estimation of the available bandwidth by the sources. This is an important advantage of window-based algorithms [5] that motivates a deeper analysis of their behaviors.

To perform such analysis, we distinguish between lossless and lossy transmission environments. By lossless, we refer to networks that are free of non-congestion-related losses. In contrast, lossy environments are subject to non-congestion-related losses, e.g. radio losses. It is worth noting that both lossless and lossy environments may thus experience congestion losses. In lossless environments, window-based protocols have been investigated in details. TCP is certainly the most well-known window-based congestion control algorithm. TCP is implicit and end-to-end in the sense that congestion in the network is inferred by the end-systems, exclusively based on the network response (e.g., packet loss and delay) [6]. It only provides a late and coarse feedback about the network status. Moreover, packet loss is known to be a poor signal of congestion, since congestion is in general not the only source of loss. Consequently, TCP becomes inefficient and prone to instability and unfairness when the delay-bandwidth product of a connection increases [7–10]. Explicit congestion control mechanisms [1,11,12], where routers provide explicit feedback to the sender regarding network congestion, have been shown to solve the problems related to imprecise and late feedbacks. These protocols are more responsive and stable than the conventional TCP, and become especially beneficial when the delay-bandwidth product increases. Hence, it is generally admitted that window-based algorithm provides a valid congestion control solution in lossless environments, especially if some explicit support from the network is provided to the transport layer.

In lossy environments, this conclusion does not hold anymore, neither for the implicit nor the explicit framework. In the implicit framework, interpreting any kind of loss as a congestion notification results in flow starvation on lossy links. This problem has been extensively studied for TCP connections in wireless environments [13–16]. These works are further presented and compared to our work in Section 9. In the explicit control framework, even if the sender adjust its congestion window independently of losses, the presence of losses is still able to strangle the connection. This is because the head of the congestion window is anchored to the last acknowledged data in conventional implementations of the packet conservation principle. As a consequence, the congestion window stays blocked by a lost packet as long the packet has not been recovered and acknowledged by the receiver. From these observations, it becomes relevant to argue about the ability of window-based protocols to support a significant amount of losses. We propose to focus on this fundamental question in this paper. To that aim, we explore how to decouple the output rate of a window-based congestion control algorithm, from losses in the network. In a sense, we provide the mean to extend window-based congestion control algorithms to lossy environments. We are thus not directly concerned by the stability and convergence issues associated to a congestion control protocol, as studied in [2,17,18]. Instead, we are interested in the design of mechanisms that permit to transpose a given stable window-based control algorithm from a lossless towards a lossy environment.

The outline of our approach can be summarized as follows. In general, the sources of inefficiency for window-based congestion control protocols in presence of non-congestion-related losses are mainly twofolds. First, the erroneous interpretation of a loss as a signal of congestion ends up in congestion window deflation. Second, the anchorage of the congestion window to the last acknowledged data segment prevents to send new data before a previously lost segment has been successfully retransmitted. This creates an indirect coupling between the occurrence of loss and the effective rate of the connection. In order to circumvent the first problem, we promote the use of explicit congestion control mechanisms. As long as explicit information about congestion is received by the sender, there is no reason for the sender to infer the congestion state from losses or delay measurements. Hence, losses are not interpreted as a congestion signal anymore, and do not cause congestion window deflation. This in turns prevents all issues related to the discrimination of congestion and non-congestion-related losses. Two explicit congestion control algorithms are considered in this paper, in order to validate the proposed loss-resilience mechanisms. The first one is the eXplicit Control Protocol (XCP), introduced by Katabi et al. [1]. The second one is a novel eXplicit TCP-like congestion control algorithm (XTCP), where routers only provide a coarse binary feedback about congestion. The second issue is addressed by maintaining the connection active and efficient during loss recovery periods. This is done by resetting the connection recovery timer each time a packet acknowledgment reaches the source. The sender window is also inflated in response to duplicate acknowledgments. In addition, a novel retransmission timer is proposed to deal with multiple losses of data segments. Based on the explicit congestion control framework, we therefore propose to implement aggressive retransmission mechanisms in parallel to the emission of novel data segments by the sender.

We rely on NS simulations to evaluate the potential of the proposed loss management and retransmission mechanisms in the explicit window-based congestion control framework. Our findings can be summarized as follows:

- In a simple, yet representative heterogeneous network topology, we observe that explicit congestion control combined with the proposed loss recovery mechanisms permits to utilize fully the bottleneck link, and provide a fair share of the bottleneck bandwidth between all connections.
- For high loss ratios or bursty loss processes, we observe that XTCP only achieves fairness between lossless and lossy connections if the sender can rely on precise feedbacks from the receiver about packets arrival.
- The strategy proposed to manage the retransmission and recovery timers in the explicit framework, combined with careful inflation and deflation of the send-out window, significantly amplifies the benefit obtained by the fast retransmit and fast recovery mechanisms used in TCP Reno [6] or NewReno [19].
- The proposed eXplicit TCP protocol is shown to fairly coexist with TCP, in a single queue of an XTCP-enabled router. It certainly represents an attractive characteristic for its deployment.

The paper is organized as follows. Section 2 introduces the framework of our study, and defines the state variables that characterize a window-based control algorithm. Section 3 then presents the XCP and XTCP explicit congestion control algorithms that are considered in this paper for validating the proposed loss-resilience mechanisms. In Sections 4 and 5, we describe how to implement loss-resilient mechanisms in the explicit congestion control framework, based on information conveyed by cumulative, or respectively precise acknowledgment packets. Sections 7 and 8 validate and compare loss-resilience mechanisms in XCP and XTCP, based on NS simulations. Finally, Section 9 puts our contribution in perspective with earlier related works.

2. Preliminaries

2.1. Terminology for window-based protocols

As the goal of our paper is to explore the ability of explicit window-based congestion control protocols to support transmission losses, we briefly recall here the state variables that characterize a window-based connection. The section follows conventional TCP terminology, so that readers that are familiar with these notions may skip the section.

In order to regulate packet transmission, the senders uses feedbacks from the receiver about the state of session. We limit our study to window-based control protocols based on positive acknowledgment (ACK), where the receiver sends feedback information in response to correctly received packets. The feedback information is based on the data sequence number and, possibly on the packet sequence number present in the packet header. The *data sequence number* associated to a packet identifies the data segment conveyed by the packet. Two transmissions of the same data are thus characterized by the same data sequence number. The *packet sequence number* identifies each packet transmitted by the sender. It corresponds to a counter incremented by one each time a new packet is sent. Two packets that (re)transmit the same data at different time instants have thus the same data sequence number, but distinct packet sequence numbers. Based on the data sequence number definition, a *cumulative acknowledgment* with a sequence number equal to N, indicates that all the data segments with a data sequence number up to and including N have been correctly received. At any time, the *lack* state variable records the largest cumulative acknowledgment ever received by the sender.

By definition, window-based congestion control limits the number of transmitted packets, which have not been acknowledged yet. This number of packets in transit between sender and receiver, is referred to as the *congestion window size*, and is generally denoted *cwnd*. This variable is the one that constraints the rate of the connection based on the network state of congestion. It is adjusted based on the information explicitly received from the routers, or implicitly inferred from the observation of the network behavior (loss and delay). Along with the congestion window, the *send-out window* denoted *swnd*, describes the data that are eligible for transmission, which are the data whose sequence number lies between *lack* and $lack + swnd$. In practice $swnd \geqslant cwnd$, and the difference between *swnd* and *cwnd* corresponds to data packets that have already left the network, and are stored at the receiver. Note that *swnd* is upper bounded by the receiver advertised window denoted *rwnd*, which reflects the receiving buffer capacity. Finally, the data sequence number of the next segment to be considered for transmission is given by the *nextseq* state variable. The strategy employed at the server, and in particular the update of *lack*, *swnd* and *nextseq* upon reception of receiver acknowledgments or timer expiration, and the monitoring of *swnd*, directly drives the behavior of the congestion control algorithm.

2.2. Overview of TCP retransmission and recovery mechanisms

Before describing in details novel mechanisms to improve window-based connection over lossy links, we present a brief overview of TCP retransmission and recovery mechanisms. Based on the assumption that the receiver sends out the largest possible cumulative acknowledgment at each packet arrival, TCP implements three mechanisms to recover from a loss. They consist in two retransmission schemes respectively initiated by a duplicate or a partial acknowledgment, and a connection reset triggered by a recovery timer. We describe below these three mechanisms, and set the framework for the description of our improved solutions. Readers that are familiar with TCP can skip the remaining of this section.

2.2.1. Retransmission based on duplicate acknowledgment

In presence of packet losses, duplicate acknowledgments may be generated in response to the reception of data that have a higher data sequence number than any data segments that have not been received yet. Upon reception of a *duplicate ACK*, the sender infers that the data immediately following the largest acknowledged data, i.e., the $(lack + 1)$th data segment, has either been delayed or lost by the network. In practice [6], the sender waits for *dupackthreshold* duplicate ACKs before it concludes that the $(lack + 1)$th data segment has been lost, and retransmits it. At the same time, the arrival of a duplicate acknowledgment at the sender indicates that a packet has reached the receiver and left the network. In accordance with the congestion window definition, the send-out window *swnd* is incremented by one and a new data packet can be sent out in response to the arrival of a duplicate acknowledgment.

2.2.2. Retransmission based on partial acknowledgment

Retransmission based on partial acknowledgment has been proposed by the NewReno version of TCP [19]. It is expected to help when multiple packets are lost from a single window of data. Among new data acknowledgments, NewReno distinguishes between *complete* and *partial acknowledgments*. Let *olack* and *nlack*, respectively, denote the largest data sequence number acknowledged before, respectively after the reception of a new ACK. A new ACK is defined to be a complete ACK if it acknowledges all the data segments that have been sent before the last (re)transmission of the $(olack + 1)$th segment. On the contrary, a new ACK is a partial acknowledgment if it only indicates the correct reception of a subset of these segments.

Partial acknowledgments are interpreted as the loss of the $(nlack + 1)$th data segment, and the sender retransmits the corresponding data unit.

2.2.3. Recovery timer

In complement to retransmission mechanisms, all TCP implementations use a *recovery timer* as a recovery mechanism of last resort. The expiration of this timer indicates that the connection stayed idle for a while, and has to be reset. A *connection reset* simply consists in setting *cwnd* to one, and *nextseq* to $lack + 1$. All counters and timers are also reset to zero.

2.3. Motivations for explicit congestion control

Despite its loss recovery mechanisms, TCP is unable to face non-congestion-related losses. The erroneous interpretation of a loss as a signal of congestion actually ends up in congestion window deflation and connection starvation. To circumvent this problem, we promote the use of explicit congestion control [1,11,12], where the sender can strictly and uniquely rely on the routers feedback to control the size of its congestion window. Hence, it does not interpret losses as a signal of congestion avoids to undertake actions to slow down the connection upon reception of duplicate ACKs. It rather tries to maintain the connection active and effective as long as (duplicate or new) ACKs are regularly received, which indicate that the connection is still alive. However, the anchorage of the congestion window to the last acknowledged data segment prevents to send new data before a previously lost segment has been successfully retransmitted. This creates an indirect coupling between the occurrence of loss and the effective rate of the connection. We present two explicit congestion control algorithms in the next section. Then, we explain in Sections 4 and 5 how to achieve that non-trivial objective of maintaining the connection when the receivers returns cumulative and explicit acknowledgments, respectively.

3. Explicit window-based congestion control algorithms

This section presents the two specific window-based congestion control algorithms considered in this paper. Both protocols rely on the explicit transmission of information about the state of congestion at the routers to compute the *cwnd* value. However, the information provided about congestion is different in both cases. The first protocol, introduced in [1], exploits an accurate feedback from the routers. In the second novel protocol, we propose to rely on a coarse binary feedback to notify about congestion. Simulations are then used to evaluate the impact of the congestion feedback granularity on the robustness of lossy connections.

3.1. The explicit control protocol (XCP)

The first protocol that we consider here is the explicit control protocol (XCP), proposed in [1]. XCP is window-based, and controls the size of the congestion window based on explicit and accurate feedback from routers. In short, XCP is based on a few bytes of control information conveyed in the packet headers. To control the link utilization, routers inform the senders about the degree of congestion in bottleneck links. In a router, the congestion information is computed based on the mismatch between the aggregate traffic rate and the link capacity, and is adjusted according to the delay expected for the feedback packet. Fairness is achieved by reallocation of bandwidth between individual flows. Extensive simulations demonstrate that XCP maintains good utilization and fairness among lossless connections, while maintaining small standing queue sizes [1]. In particular, it has been shown that XCP outperforms TCP when the per-flow delay-bandwidth product becomes large. In Section 7, we consider the combination of XCP with the proposed loss recovery mechanisms.

3.2. An explicit TCP (XTCP)

As an alternative to XCP, we propose a new explicit congestion control protocol named explicit TCP (XTCP). Overall, the difference between the proposed explicit TCP and conventional TCP is the method to infer congestion from the network feedback. The TCP sender implicitly infers congestion from a lost data [20]. On the contrary, XTCP requires an explicit feedback from the routers to infer that congestion occurred, and then decreases its congestion window. In order to decouple the gain provided by an explicit framework, from the congestion control mechanism itself, XTCP mimics the TCP behavior. It relies on a minimalist binary feedback from the routers (just as TCP relies on the binary congestion signal inferred from losses), and adopts the exact same additive increase

– multiplicative decrease behavior as TCP. Hence, the XTCP sender just probes the network to the point of congestion before backing off, just as TCP would proceed. A TCP-like explicit protocol is not only interesting for comparison between explicit and implicit congestion controls, but obviously presents also advantages in terms of deployment issues (see Section 8.1).

In more details, the binary feedback is provided by a *congestion flag* contained in the XTCP packet header. The flag is initialized to zero by the sender, and is set to one when the packet encounters a congested router. When the packet reaches the receiver, the flag is copied in the ACK header, and returned to the sender. Upon ACK reception, the sender decides whether the congestion window should be decreased or increased based on the congestion flag. Similarly to TCP, the congestion window is incremented each time an ACK with a null congestion flag is received, and divided by two when the sender infers a congestion event based on the returned congestion flags. By definition, a congestion event occurs when an ACK with a congestion flag set to one is received, and when the latest congestion event is older than one RTT. This is to avoid multiple backoffs during one RTT. In practice, an exponential weighted average of the RTT samples is used to estimate RTT.

We now explain how routers define the congestion flag. Formally, a *congestion counter* is associated with every queue in the network. Each time a queue drops a packet due to congestion, the congestion counter associated to the queue is incremented by one. When an XTCP packet leaves the queue to be sent out to the output link, if the counter is positive, the congestion flag of the XTCP packet is set to one, and the counter is decremented by one.[1] The packet whose congestion flag is set to one does not necessarily belong to the same flow as the packet whose drop is responsible for incrementing the congestion counter. There is no need to maintain per-flow congestion states in the router. Note finally that XTCP does not make any assumption about the queue management strategy used in routers. In our simulations, XTCP has been tested both with Droptail or RED policies [21].

[1] To make sure that we do not run into a situation where the counter is positive, and the queue is empty, we only increment the counter if its current value is smaller than the number of packets present in the queue.

4. Loss recovery based on cumulative acknowledgments

This section proposes original retransmission and send-out window management mechanisms to recover from losses while preserving connection efficiency, in the context of an explicit congestion control framework. In summary, connection efficiency is preserved by (i) partial deflation of the send-out-window upon reception of a new ACK, (ii) reset of the recovery timer in response to any ACK, and (iii) definition of a novel retransmission timer. We describe these three mechanisms and explain how they interact and complement each others. As a main outcome, we show that the introduction of a new *retransmission timer* induces significant changes in the behavior of the retransmission and recovery mechanisms defined for TCP and generally accepted in the context of implicit congestion control. The retransmission timer offers an efficient strategy to handle multiple losses of a segment and enables more frequent resets of the recovery timer. This in turn increases the benefit obtained from a careful management of the inflation of the send-out window in presence of losses.

4.1. Partial deflation of the send-out-window

As explained in Section 2.2, the arrival of a duplicate acknowledgment at the sender indicates that a packet has reached the receiver and thus left the network. Hence, the send-out window *swnd* in TCP is progressively and artificially inflated upon reception of duplicate ACKs. Upon reception of new data acknowledgment, the head of the send-out window is moved to the largest acknowledged data segment. It possibly oversteps a number of data segments whose earlier receptions have triggered duplicate ACKs, and caused inflation of *swnd*. In order to ensure that the number of packets in transit is equal to the congestion window, *swnd* has thus to be decremented. We propose to decrease its value according to the number of segments that have triggered duplicate ACKs in the past, but which are implicitly acknowledged by the reception of a new ACK. In other words, *swnd* is decremented by $nlack - (olack + 1)$ in the explicit congestion control framework in order to preserve connection efficiency while recovering from losses. In contrast, *swnd* is simply reset to *cwnd* upon reception of a new ACK in the conventional implementation proposed by TCP Reno [6]. This is because the *cwnd* back off anyway alleviates the potential advantage taken from a partial deflation of *swnd* in an implicit congestion control environment.

4.2. Aggressive reset of the recovery timer

In an explicit congestion control framework, duplicate ACKs should not be interpreted as a signal of congestion. They indicate that the connection is alive, and even convey fresh information about congestion state. It is therefore important to keep the connection active upon reception of a duplicate ACK and to postpone the expiration of the recovery timer. Hence, we propose to reset the recovery timer both in response to a new ACK or a duplicate ACK when the congestion control is explicit. In contrast, TCP does not reset the recovery timer upon reception of a duplicate ACK that does not cause a retransmission. Note that, even if aggressive resets of the recovery timer ensure that the connection is maintained, the efficiency of that strategy strongly depends on the careful management of the temporary inflation of the send-out window described above. Both mechanisms closely interact and complement each others.

4.3. Retransmission based on a timer

The partial window deflation and the aggressive timer reset described above contribute to maintain the connection active while conventional TCP mechanisms deal with retransmission of lost packets. However, it is well-known from the TCP literature that triggering retransmissions based on duplicate or partial ACKs is unable to deal with multiple losses of the same segment, typically because the *dupacks* state variable is only reset to zero after the acknowledgment of new data[2].

In an explicit congestion control framework, we however claim that the recovery timer should be reset at every duplicate ACK, and that the connection efficiency should be preserved by progressive inflation of the send-out window. In these conditions, the multiple losses of a data segment result in a situation where *swnd* goes to infinity (or at least to *rwnd*), before the recovery timer gets the opportunity to expire and causes the retransmission of the lost segment. To circumvent the problem, we propose to implement a retransmission mechanism based on a novel *retransmission timer*. When the timer expires, the $(lack+1)$th data segment is retransmitted. The retransmission timer is reset every time new data are acknowledged, and every time a data segment is retransmitted, but is not reset upon reception of a duplicate ACK that does not cause retransmission.

In such a mechanism, the retransmission timout needs to be defined carefully. A short timeout results in fast retransmission, and rapid loss recovery. However, one should avoid to trigger retransmissions for packets that are still in transit. Stability becomes an issue when retransmissions are based on a timer. A bad choice of the timeout value might cause the sender to inject a new packet into the network before an old one has exited. This violates the ‘packet conservation principle’ that guarantees stability for window-based transport protocols [4]. In order to avoid instability, we propose to choose the retransmission timeout larger than a conservative estimation of the round trip time. Specifically, in our simulations, the retransmission timeout has been defined twice as large as the recovery timeout that represents a conservative estimation of the round trip time (see Section 2.2). Here, it is worth noting that choosing a retransmission timeout that is larger than the recovery timer guarantees that the connection ends up in a recovery phase and does not swamp the network with inadequate retransmissions, even in case of underestimation of the round trip time. We have also observed in the simulations presented in Sections 7.1 and 8.2 that the expiration of such a retransmission timer is always appropriate. The system is stable even when the round trip time is under-estimated (as well as the recovery and retransmission timers). Moreover, our simulations have revealed that the behavior of the loss-resilience system is not sensitive to the actual value of the retransmission timeout.[3] Hence, we conclude that defining the retransmission timeout as a roughly proportional but larger version of the recovery timeout prevents unstable behavior of the system. At the same time, it permits efficient recovery of multiple losses of the same packet.

From a functional point of view, it is worth noting that the retransmission timer triggers the

[2] If *dupacks* is reset to zero immediately after the retransmission of a packet, subsequent duplicate acknowledgments that correspond to the same window of emission would trigger an additional and probably useless retransmission.

[3] Setting the retransmission timeout to 1.5 or 3 times the recovery timeout did not significantly affect the behavior of the system.

required retransmissions before expiration of the recovery timer, even if its timeout is larger than the recovery timeout. This is because the explicit congestion control authorizes frequent resets of the recovery timer each time a new or duplicate ACK is received, while it only resets the retransmission timer in response to a new ACK. That complementarity between the recovery and the retransmission timer is fundamental and probably represents one of the most important findings in our study. We show later in Sections 7.1 and 8.2 that the proposed retransmission timer improves the connection efficiency significantly beyond the benefit that is already provided by partial deflation of *swnd* and aggressive reset of the recovery timer.

4.4. Summary and discussion

The main differences between the retransmission and recovery mechanisms proposed in this paper and the ones implemented in TCP are twofold. First, losses are not interpreted as a signal of congestion in the explicit control framework, which avoids the need to sharply reduce the congestion window in response to duplicate ACKs. It permits to keep the connection active and efficient as long as enough ACKs are received. The connection efficiency is preserved since the congestion window stays unchanged upon reception of a duplicate ACK, and the send-out window is only partially deflated upon reception of a new ACK. Second, the presence of a retransmission timer also contributes to preserve the connection efficiency as it allows for a reset of the recovery timer each time a new ACK or a duplicate ACK is received. Such an aggressive reset strategy is especially beneficial in conjunction with the partial deflation of the congestion window. Hence, we conclude that the window and timer management mechanisms introduced above nicely complement each others. They contribute together to the robustness of the explicit congestion control framework in the presence of packet loss.

Finally, we emphasize that the proposed mechanisms are strictly dedicated to an explicit congestion framework, but can not contribute to improve the performance of an implicit congestion control algorithm. As an example, our simulations have revealed that the partial deflation of *swnd*, already proposed by TCP NewReno, brings a significant benefit when implemented in the context of an explicit congestion control, but does not significantly help in a classical TCP context. In TCP, duplicate ACKs cause a decrease of the congestion window size, which alleviates the benefit obtained from a partial deflation of *swnd*. Similarly, in an implicit congestion control framework, it is far better to reset the connection in presence of multiple losses, rather than to use a retransmission timer that keeps the connection alive but severely strangled due to the *cwnd* back off.

5. Loss monitoring and recovery with explicit acknowledgments

With the limited information available from cumulative acknowledgments, a sender can only learn about a single lost packet per round trip time. We rather consider here precise feedbacks about packet arrival, and analyze how the window-based congestion control can take advantage of such rich feedbacks in lossy environments. We first present the protocols that allow to include a rich feedback within receiver acknowledgments. Then we describe the novel loss retransmission mechanisms that have been specifically designed to exploit this feedback information in an explicit congestion control framework. Simulation results are later provided in Sections 7 and 8.

5.1. Packet sequence number feedback

The main limitation of cumulative acknowledgments comes from the imprecise information that they convey about the status of the connection. Upon reception of a duplicate ACK, the sender can not infer which exact packet has triggered the ACK, and consequently, which data segment has reached the receiver. A simple way to circumvent this limitation is to uniquely identify every packet that is sent on the network, and to force the receiver to include that unique identifier within the acknowledgments returned to the sender. In our simulator, this has been done by adding a field to the packet header that contains its packet sequence number. Remember from Section 2 that the packet sequence number is defined based on a counter incremented by one each time a new packet is sent. Every time the source sends a packet, it writes the state of the counter in the packet header, and increments the counter by one. This concept of packet sequence number has been introduced previously by Keshav and Morgan [22] for the design of efficient retransmission mechanisms in the context of rate-based congestion control, where the transmission of new packets and the loss recovery mechanisms are totally decoupled [22]. On the contrary, we are interested in window-based congestion control. We analyze the benefit offered by a feedback including packet numbers when the emission of novel data segments is directly constrained by a send-out window, whose head is attached to the largest cumulative acknowledgment received by the sender.

Note that another way for the sender to learn about the data segments that have reached the receiver is the selective acknowledgment (SACK) option proposed for TCP [23]. We have not consider this mechanism here, but we expect that conclusions drawn from our implementation can be extended to SACK implementations.

5.2. Retransmission based on accurate loss monitoring

When informed about the data segments that have been correctly received, the sender can adopt intelligent strategies to retransmit the missing data. Based on the feedback about the packets that have been received in or out of order, the sender updates a *loss-monitoring window*. It records information about the segments that are still waiting for a cumulative ACK, and its size is limited by the largest number of out-of-sequence packets that can be buffered at the receiver (i.e., *rwnd*). We now define how the loss-monitoring information is maintained, and later explain how this information is exploited to trigger data retransmissions.

Let $N < rwnd$ denote the size of the loss-monitoring window, in packets. Given the largest acknowledged data sequence number ($lack$), the loss-monitoring window stores the state of all segments whose data sequence number j verifies $lack < j < lack + 1 + N$. In practice, we use a circling buffer to store the state of the relevant data segments. Let $W[\cdot]$ be an array of size N. At any time, $W[j \bmod N]$ stores the loss-monitoring window state corresponding to the jth data segment. Given $lack < j < lack + 1 + N$, $W[j \bmod N]$ is defined as follows:

- $W[j \bmod N] = FREE$, with $FREE$ being a constant flag value, when the jth data segment has not yet been sent over the network;
- $W[j \bmod N] = RECV$, with $RECV$ being a constant flag value, when the jth data segment has been received out of order by the receiver;
- $W[j \bmod N] = X$, with $X > 0$ being the packet sequence number of the latest packet sent over the network and conveying the jth data segment, in any other case.

In more details, the loss-monitoring window state variable is maintained as follows:

1. First, each array position is initialized to the constant $FREE$ value, indicating that each block of the array is available to store the state of future data segments.
2. When a packet is sent out, the loss-monitoring window is updated as follows. Let d denote the data sequence number of the data segment conveyed by the packet, and p be the packet sequence number. Then, $W[d \bmod N]$ is set to p, indicating that all packets with a packet sequence number larger than p have been sent out before the last emission of the dth data segment. Recording this information is important for the retransmission mechanism proposed hereunder.
3. Upon reception of a new ACK, the *lack* state variable is updated. $W[j \bmod N]$ is reset to $FREE$, $\forall j$ such that $olack < j < nlack$. This indicates that the corresponding positions of the array are now available to store the state of future data segments. Note that before sending out a new data segment n on the network, the sender has to check that $W[n \bmod N] = FREE$, to ensure that the new segment will not exceed the storage capacity of the loss-monitoring window.
4. Upon reception of a duplicate ACK, $W[d \bmod N]$ is set to $RECV$, where d denotes the sequence number of the data segment conveyed by the packet that has triggered the ACK. It means that the dth data segment has been correctly received. Note that packet number can be read from the header of the received ACK, and that the sender obviously knows which data segment has been sent in a given packet.

Given the state of the loss-monitoring window $W[\cdot]$, the design of novel retransmission mechanisms is driven by the following rules. First, a data segment can only be retransmitted once the sender has received *dupackthreshold* acknowledgments, triggered by packets that have been sent out later. Second, the number of retransmissions per *dupackthreshold* acknowledgments is limited to one. Third, a data segment should only be retransmitted once per RTT, so as to preserve the 'packet conservation' principle. To follow these rules, we define the state variable *rseqn* to denote the sequence number of the data segment that is expected to be the best candidate for a retransmission. Among all data segments monitored in the window W, *rseqn* is defined as the segment with the least recent (re)transmission, and for which the sender has no indication about correct delivery. Formally, given $W[.]$, *rseqn* is defined by

$$rseqn = \operatorname*{arg\,min}_{lack+1<j<lack+1+N,\ W[j]\neq RECV/FREE} W[j]. \tag{1}$$

Let *dupcount* then denote a counter that is incremented by one every time an ACK triggered by a packet whose sequence number is larger than $W[rseqn]$ reaches the sender, without indication on the correct reception of the *rseqn*th data segment. This happens when the ACK or packet corresponding to the last retransmission of the *rseqn*th data segment has either been delayed or lost. When *dupcount* reaches *dupackthreshold*, the *rseqn*th data segment is finally retransmitted. After a retransmission, or after the sender received an ACK indicating the correct reception of the *rseqn*th data segment, *dupcount* is reset to zero and *rseqn* is updated based on Eq. (1). To avoid multiple retransmissions of the same data in a single RTT, *dupcount* is only reset to zero once the packet sequence number gets larger than $W[rseqn] + cwnd$.

We note here that the send-out window is respectively inflated or partially deflated in response to a duplicate ACK or to a new ACK, as explained in Section 4.1. The inflation/deflation process is important to keep the connection active while lost segments are retransmitted. Moreover, both the retransmission timer and the aggressive recovery timer defined in Sections 4.3 and 4.2 are also used in conjunction to the loss-monitoring window.

6. Simulation setup and performance evaluation

Sections 7 and 8 analyze through simulations the advantages and limitations of the retransmission mechanisms presented in Sections 4 and 5, in the context of explicit window-based congestion control. The purpose of those simulations is to check that the proposed loss recovery mechanisms permit to exploit a stable window-based control algorithm in a lossy environment. We are thus not directly concerned by the stability and convergence issues – we assume they have been studied while designing and testing the protocol in lossless environments – but rather by the preservation of the protocol efficiency in presence of losses, as attested by a fair partition of bandwidth between contending lossy and lossless connections. We present results based on ns-2 simulations in a simple topology that has been chosen in order to check whether the fair control of the connections is preserved in presence of losses. This topology is represented in Fig. 1, where n sources share a bottleneck link. Half of the flows ends up in node N2, through a loss-free link, while the other half ends up in node N3, across the lossy link. We refer to the flows that go through the lossy (loss free) link as lossy (loss free) flows. All links have a bandwidth equal to 5 Mbps, and a propagation delay of 20 ms. In the rest of the paper, the number of sources n is equal to 10, packet size is set to 1000 bytes, and losses generated on the lossy link are either randomly distributed (default case), or follow a bursty process.

In order to evaluate the performance of the proposed loss-resilience mechanisms, we measure the bottleneck link utilization and analyze how the bottleneck bandwidth is partitioned between loss-free and lossy links. A desirable solution ensures full-link utilization and a fair partition of the bottleneck between the connections, independently of whether they are affected by losses or not. Hence, for the above topology, the performance of loss-resilient protocols is estimated by comparing the sum of the bottleneck throughputs measured for lossy and loss-free flows. An equal usage of the bottleneck by lossy and loss-free flows reflects good performance of the loss-resilience mechanisms.

To support the reading, Table 1 introduces the terminology used to denote the transport protocols obtained when combining XCP [1] or XTCP with several loss retransmission mechanisms. In Table 1, XCP_dumb refers to the implementation proposed in [1]. It simply relies on the mechanisms implemented by TCP Reno to recover from losses, without exploiting the advantages provided by the explicit congestion framework. In contrast, XCP does not halve *cwnd* upon reception of a duplicate

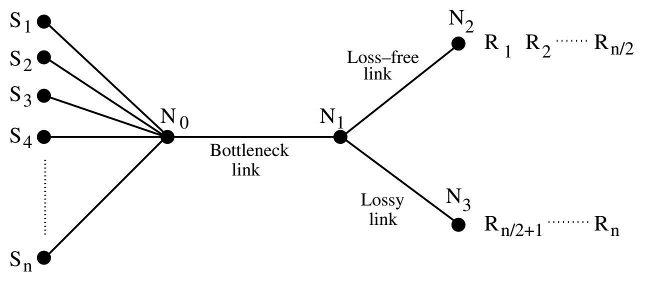


Fig. 1. Network topology reflecting different users accessing a bottleneck through links with different loss characteristics, e.g. wired and wireless.

Table 1
Acronyms for loss-resilient XCP and XTCP protocols

Acronym	Definition
XCP_dumb	eXplicit Control Protocol with retransmission and recovery mechanism implemented as in TCP Reno, similar to [1]
XCP	eXplicit Control Protocol with partial deflation of *swnd* and preservation of *cwnd* upon loss observation (see Section 4.1)
XCP_PA	XCP + retransmissions based on partial ACKs (see Section 2.2.2)
LR-XCP	XCP + retransmission timer (see Section 4.3)
LR-XCP_PA	XCP_PA + retransmission timer
LR-LMXCP	XCP + retransmissions based on accurate loss-monitoring (see Section 5)
LR-XTCP	eXplicit TCP with retransmission timer and recovery mechanisms adapted to the explicit congestion control framework (see Section 4.4)
LR-XTCP_PA	LR-XTCP + retransmissions based on partial ACKs (see Section 2.2.2)
LR-LMXTCP	eXplicit TCP + retransmissions based on accurate loss-monitoring (see Section 5)
LR-LMXTCP-TCPfriend	LR-LMXTCP + mechanism to ensure TCP friendliness (see Section 8.1)

ACK, and only partially deflates *swnd* upon new ACK reception. We then use the prefix LR to denote the implementations of retransmission timer and aggressive reset of the recovery timer (see Sections 4.3 and 2.2.3). The suffix PA then indicates that partial acknowledgments are used to trigger fast retransmissions (see Section 2.2.2). Finally, LMXCP assumes that the packet sequence number is conveyed by the packet header to allow for accurate loss monitoring, as described in Section 5.2. The last acronym, namely LR-LMXTCP_TCP-friend, refers to a TCP-friendly version of LR-LMXTCP, defined in Section 8.

7. Loss-resilient XCP validation

7.1. Performance of loss-resilient mechanisms

As explained above, the performance of the proposed loss recovery mechanisms can be evaluated in terms of the fairness between the lossless and lossy connections that share a common bottleneck link. Therefore, we first analyze the throughputs offered to lossy and lossless flows in the network topology presented in Fig. 1. On the one hand, Fig. 2a illustrates the problem encountered by the reference implementation of XCP [1] in lossy environments. We observe that the presence of losses causes starvation of the lossy flows. This is because, by default, the XCP loss recovery mechanisms are the ones optimized for TCP, and do not exploit the explicit frame-

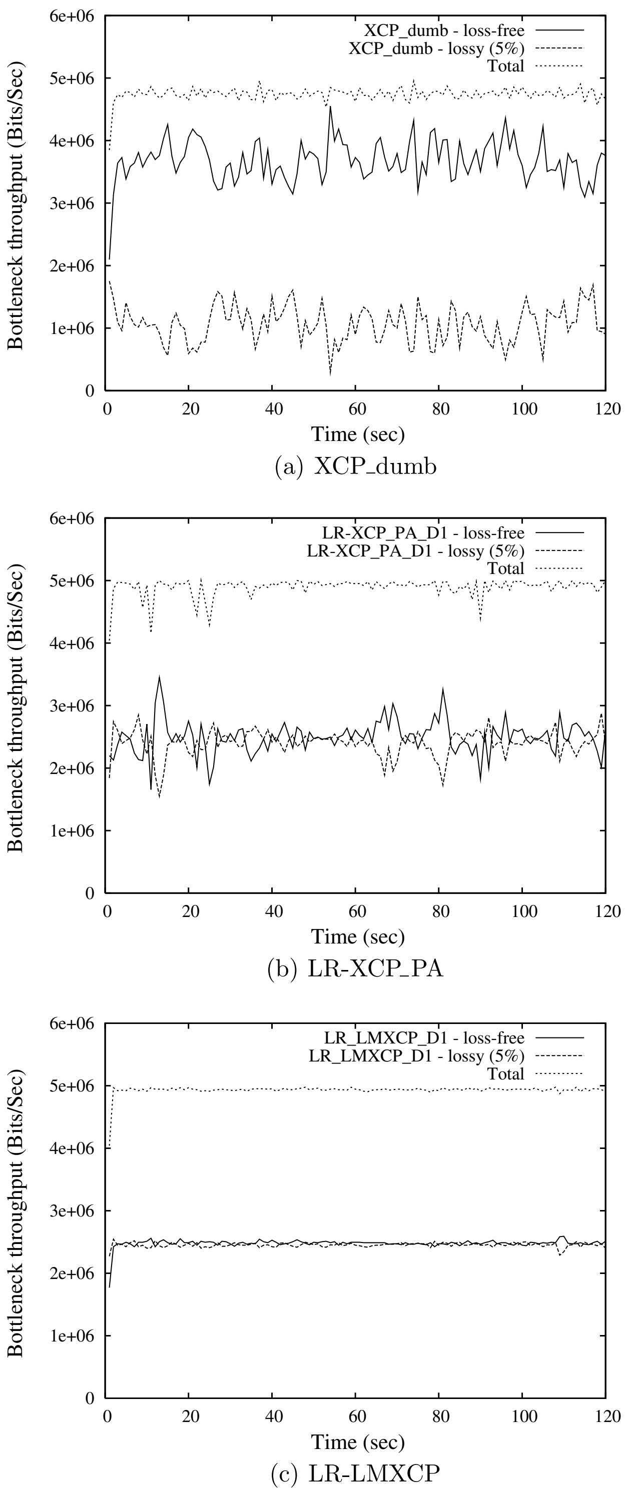


Fig. 2. Sums of throughputs measured respectively for loss-free and lossy flows. Sums of throughputs are plotted as a function of time, and loss rate is set to 5% for lossy connections. In all graphs, the *dupackthreshold* parameter is set to 1 (as indicated by D1).

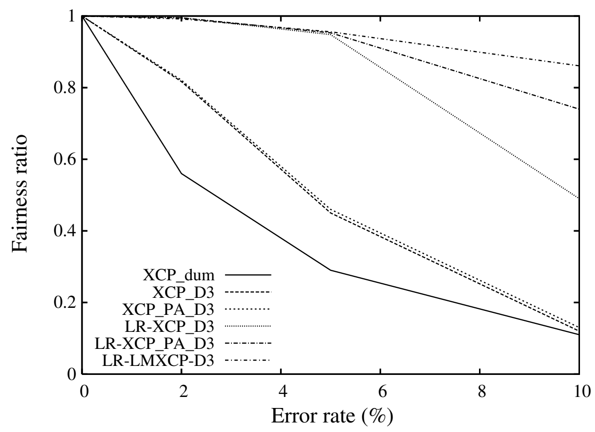


Fig. 3. Fairness ratio measured between lossy and loss-free flows as a function of the loss rate. *dupackthreshold* = 3.

work specificities. On the other hand, we show in Fig. 2b and c, that the presence of loss-resilient mechanisms dedicated to the explicit control framework permits a better distribution of resources between heterogeneous connections and significantly improves the fairness in comparison with Fig. 2a. We also observe that the accurate monitoring of losses performed by LR-LMXCP mitigates the throughput fluctuations due to loss-related connection backoffs.

We now analyze in more details the performances of the proposed algorithms. Fig. 3 presents the fairness ratio measured between lossy and loss-free flows as a function of the loss rate. The fairness ratio between lossy and loss-free flows is defined as the ratio between the sums of throughputs measured respectively for all lossy and loss-free flows on the bottleneck link. Without surprise, we observe that the fairest and the worst bottleneck bandwidth allocation are achieved by the LR-LMXCP and the XCP_dumb protocol, respectively. In addition, we observe that LR-XCP_PA performs significantly better than XCP_PA. We conclude that the presence of a retransmission mechanism based on a timer is worthwhile. By comparing LR-XCP with LR-XCP_PA, we observe that partial acknowledgments mainly help at high loss rates, i.e., when more than one packet is likely to be lost in a single RTT. Additional results presented in [24] show that the benefit of LR-LMXCP over LR-XCP is exacerbated when the *dupackthreshold* parameter is small and the loss rate is high. In that case, retransmissions are rapidly triggered by LR-LMXCP, and losses are rapidly recovered. If needed, several different data segments might be retransmitted in a single round trip time. On the contrary, even with a small *dupackthreshold*, LR-XCP can only consider the retransmission of the $(lack + 1)$th segment, and is therefore limited to a maximum of one retransmission per round trip time.

7.2. Bottleneck link utilization

The (loss-resilient) XCP protocols therefore fail to achieve full-link utilization. In Fig. 2b, we observe that the total throughput traces do not saturate at 5 Mbits/s. This is confirmed by a detailed analysis provided in [24]. When the loss rate increases, the sum of the loss-free and lossy throughputs becomes smaller than 5 Mbits/s. We explain this link utilization deficiency by the small queue sizes maintained by XCP routers [1], which makes them unable to absorb rate fluctuations that can be due to a recovery phase caused by packet losses.

In order to improve the link utilization, we propose to maintain non-zero persistent queues in XCP routers. Therefore, we have modified Eq. (1) in [1] so that the efficiency controller targets both maximal link utilization and a non-zero persistent queue. Specifically, Q is replaced by $(\overline{Q} - \gamma)$ in Eq. (1) of [1], where γ denotes the size in packets of the targeted persistent queue. $\overline{Q}$ is defined based on the Q samples as follows. In [1], a Q sample corresponds to the minimum queue seen by an arriving packet during the last propagation delay. This definition results in large fluctuations of Q along the time. To derive a stable signal from Q, we define $\overline{Q}$ as the exponential weighted average of the Q samples, i.e., each time a new Q sample is generated, $\overline{Q}$ is set to $\beta\,\overline{Q} + (1-\beta)\,Q$. In our simulation, β is set to 0.9, while the persistent queue γ is set to 10 packets. The thresholds defining the RED queue policy [21] have been increased accordingly. Fig. 4 presents the results obtained with and without persistent queues in XCP routers for the LR-XCP_PA protocol. We observe that the presence of persistent queues in routers preserves the bottleneck link utilization. We conclude that, even when accurate congestion control is possible, persistent queues in routers remain useful to absorb the unpredictable throughput fluctuations resulting from packet losses, which may cause the expiration of the recovery timer, for example.

7.3. Receiver buffers

Fig. 5 illustrates the impact of the constraint imposed on the sender by the receiver advertised window, which reflects the receiver buffer capacity,

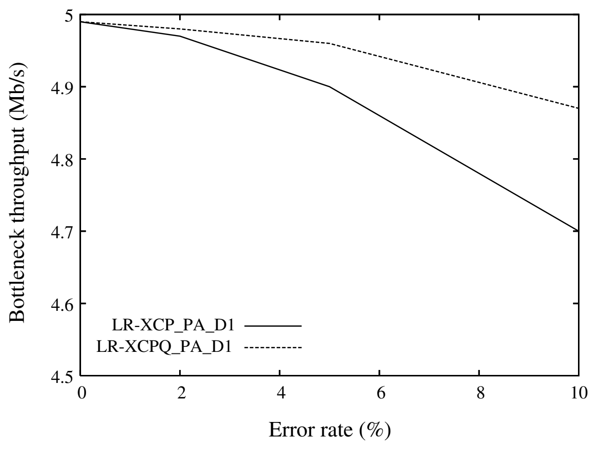


Fig. 4. Comparison between an XCP router that minimizes the persistent queue, and an XCP router that targets a persistent queue of 10 packets (LR-XCPQ_PA). The graph plots the bottleneck link utilization as a function of the loss rate for both systems.

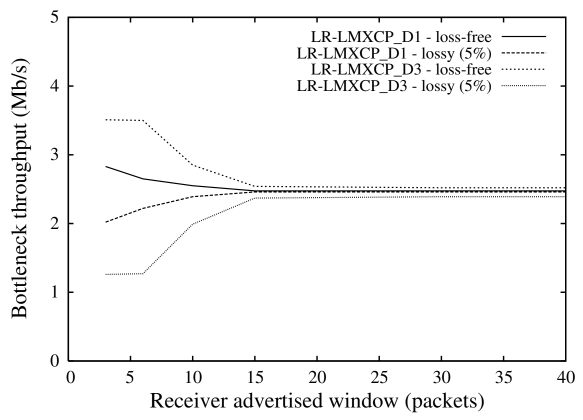


Fig. 5. Impact on fairness of the constraint imposed on the send-out window by the receiver advertised window. *dupackthreshold* is respectively set to one and three, and the lossy connections experience 5% of losses.

and its ability to store out-of-sequence packets [20]. It constrains the send-out window of the sender and limits the number of packets the sender can send in advance, while waiting for the recovery of a lost packet. Fig. 5 shows that the performance of LR-LMXCP only significantly degrades when the constraint on the send-out window becomes of the same order of magnitude as the congestion window. The same conclusion holds for LR-XCP [24]. This observation is important because it demonstrates that efficient loss-resilient mechanisms do not require large buffering capabilities from the end-hosts.

7.4. Bursty losses

Finally, we discuss the effects of bursty loss processes on the congestion control performance. Fig. 6

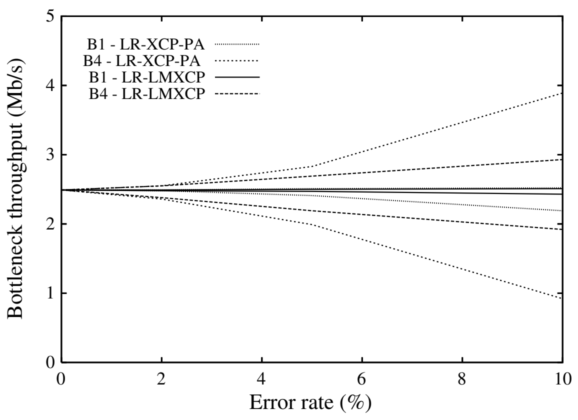


Fig. 6. Impact on fairness of the bursty nature of losses appearing on the lossy link. The acronym BX, with $X = 1$ or 4, means that each time a loss event happens, X consecutive packets are dropped on the link. The loss rate refers to the product of loss event with the X parameter. Curves on the upper (lower) half of the graph refer to loss-free (lossy) connections.

analyzes how the correlation of losses affects the retransmission mechanisms. In these simulations, each loss event causes the loss of X consecutive packets on a flow, with X ranging from 1 to 4. As expected, we observe that LR-XCP mechanisms are more sensitive to bursts of losses than LR-LMXCP. This is because LR-XCP mechanisms retransmit at most one packet per round trip time, and are thus less efficient than LR-LMXCP when multiple losses occur in the same window of data. Interestingly, we finally observe that the retransmission mechanism based on cumulative ACKs remains competitive (compared to LR-LMXCP) at low loss rates when losses are bursty.

8. Loss-resilient XTCP analysis

In this section, we explore the behavior of the novel explicit TCP (XTCP) in presence of losses, and consider its gradual deployment. We use the simulation setup described in Section 6, and analyze the combinations of XTCP with loss-resilience mechanisms that are defined in Table 1.

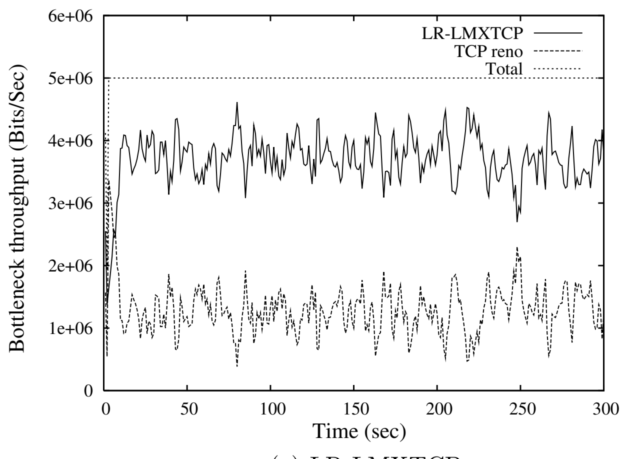
8.1. Gradual deployment: joint TCP and XTCP queuing

We consider the coexistence of TCP and XTCP traffic, and we describe a mechanism that allows end-to-end loss-resilient XTCP flows to compete fairly with TCP flows. This mechanism provides a possible path for incremental XTCP deployment.

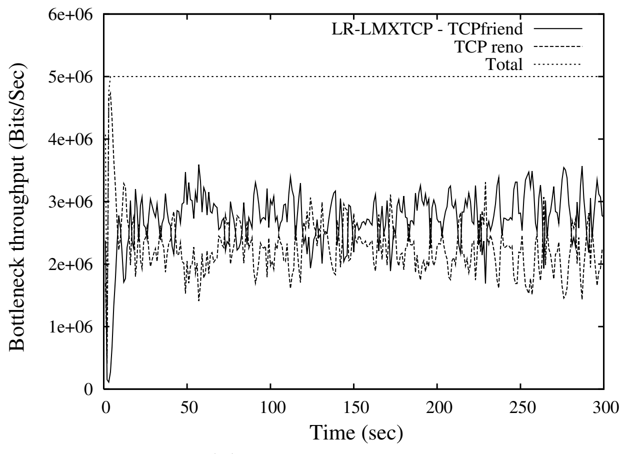
An XTCP-enabled router queues both TCP and XTCP traffics together in a single buffer, but only increments or decrements the XTCP congestion counter when it deals with XTCP packets. TCP and XTCP routers are therefore very similar, and only minor changes are necessary to enable XTCP in a router, which surely facilitates the deployment of XTCP. To start a loss-resilient XTCP connection, the sender then has to check whether the receiver and the routers along the path are XTCP enabled. As mentioned in [1] while evoking the possibility of jointly handling TCP and XCP in two distinct queues of a single router, this kind of verification can be done using TCP and IP options. If routers are not compliant with XTCP, the sender reverts to TCP.

In order to permit such a gradual deployment of XTCP, we now extend the design of loss-resilient XTCP into a TCP-friendly version, which leads to a fair allocation of resources between TCP and XTCP flows. First, we have estimated the level of (un)fairness between TCP and XTCP, based on a simulation that compare the sum of throughput measured on the bottleneck link, for TCP and LR-LMXTCP flows respectively. The topology considered for this simulation is the one described in Fig. 1. Routers obey a drop tail policy, and are XTCP enabled. All links are loss-free. Half of the flows are TCP flows. The others are LR_LMXTCP. In Fig. 7a, we observe that LR-LMXTCP flows achieve three to four times larger throughputs than TCP flows. This unfairness is mainly due to the different behavior of LR-LMXTCP and TCP when they face (congestion) losses. LR-LMXTCP handles losses in an efficient way, while TCP generally resorts to a recovery phase when more than one loss occur in a single flight of packets [6].

To increase fairness between XTCP and TCP, we propose a simple change to the design of the loss-resilient XTCP sender, so that it triggers an artificial connection back-off when it detects conditions for which TCP is expected to experience a timeout. We use the LR-LMXTCP_TCPfriend acronym to refer to this version of LR-LMXTCP. The artificial back-off emulates the TCP recovery process described in Section 2.2.3. It consists in resetting the congestion window to one, but without resetting the *nextseq* state variable to $lack + 1$. To mimic the TCP recovery phase, a state variable, denoted *recphase*, is updated to the largest data sequence number ever sent out by the sender, and the congestion window is not incremented based on received



(a) LR-LMXTCP



(b) LR-LMXTCP-TCPfriend

Fig. 7. Throughput traces showing how loss-resilient XTCP competes with TCP. (a) LR-LMXTCP is unfair to TCP, (b) LR-LMXTCP with artificial backoff simulations achieves improved fairness.

acknowledgments as long as the *lack* state variable remains smaller than *recphase*. The artificial back-off is triggered when (i) a congestion flag is received and (ii) either the congestion window is smaller than *dupackthreshold*, or two congestion flags are received in less than one RTT. These conditions reflect the fact that a loss ends up in a recovery phase for TCP either when the connection can not enter a fast recovery phase, or when two packets are lost in a single RTT. The plots provided in Fig. 7b reveal that this modification is efficient, since LR-LMXTCP_TCPfriend now competes fairly with TCP, while reaching full-link utilization.

For the sake of completeness, we discuss here cases where loss-resilient transport protocols are desirable, thereby motivating the deployment of our proposed solution. In general, we distinguish two kinds of communication applications, depending on whether they support transmission errors or not. For example, a file transfer application requires a reliable connection, while a video streaming application can support moderate loss rates. In current networks, TCP is commonly used for reliable transfers, thereby forcing every link in the network to be reliable. In particular, in wireless environments, the fact that TCP does not support non-congestion-related losses forces the network to handle radio losses at the link layer, by adjusting wireless transmission power and local retransmission mechanisms. A non desired consequence is the fact that, independently of the transport protocol (UDP or TCP), the losses affecting a video streaming application are also handled at the link level. A more desirable solution would handle losses and decide about the relevance of retransmissions at the application level, by taking into account the relative importance of packets with respect to the reconstructed video quality. The implementation of our proposed solution changes the global picture. Indeed, it gives the transport layer the ability to deal reliably and efficiently with losses that are not due to congestion. This in turns permits the wireless link to relax the constraint imposed on retransmission mechanisms, thus pushing the decisions about retransmission towards higher layers of the protocol stack and closer to the application layer. Hence, differentiated processing of losses can be implemented for distinct applications, and globally more efficient usage of wireless resources could hopefully be achieved. Typically, file transfer applications would rely on XTCP to achieve reliability on top of error-prone wireless links, while video streaming applications would implement their own loss-resilience mechanisms on top of a UDP session whose rate could be controlled based on the explicit notifications received from routers.

8.2. Loss-resilient XTCP performances

We now use simulations to analyze the performance of loss-resilient XTCP protocols. Fig. 8 presents the results in terms of fairness ratio between lossy and lossless flows. We observe that TCP rapidly starves the lossy flows. This is illustrated for TCP Reno, but the conclusion holds for other versions of TCP (New Reno and SACK), since their behavior in presence of losses is dominated by *cwnd* shrinkage upon loss detection. We also see that the retransmission based on partial ACKs only improves LR-XTCP beyond a sufficient loss rate.

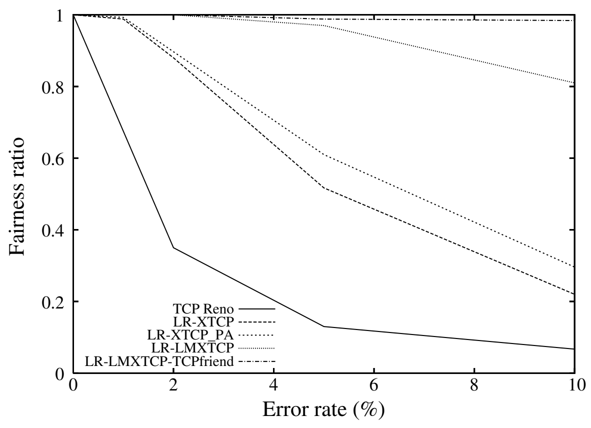


Fig. 8. Fairness ratio measured over the bottleneck between lossy and loss-free flows, as a function of the loss rate. Losses are randomly distributed. The *dupackthreshold* parameter is set to 3.

Moreover, we note that an accurate feedback, as explored by LR-LMXTCP, brings a significant improvement over approaches that are based on cumulative ACKs. Because LR-LMXTCP preserves high efficiency at high loss rates, we conclude that window-based congestion control protocols, when coupled with a precise feedback from the receiver, are able to support lossy environments. In addition to these general conclusions, we also observe that LR-LMXTCP-TCPfriend performs even better than LR-LMXTCP, since the artificial backoff introduced in this TCP friendly version improves the fairness between lossless and lossy flows.

Interestingly, in-depth comparisons between Figs. 8 and 3 reveal that LR-XTCP performs worse than LR-XCP for loss rates larger than 1%. We explain that observation by the fact that a connection controlled based on a binary congestion feedback is severely impaired when simultaneously affected by losses and congestion notifications. Indeed, upon congestion notification, *cwnd* is halved by two. In presence of losses, the head of the send-out window stays blocked at the last acknowledged data segment, and the amount of transmitted data per RTT gets directly penalized by the sharp reduction of *cwnd*. We conclude that the smooth regulation of *cwnd* supported by XCP is helpful in presence of heavy loss rates, at least when the sender infers losses based on cumulative acknowledgments. In contrast, when the sender receives accurate information about received data segments from the receiver, we observe that similar loss resilience is achieved whatever the smoothness of the congestion control mechanism, and LR-LMXCP and LR-LMXTCP perform equally well. In presence of

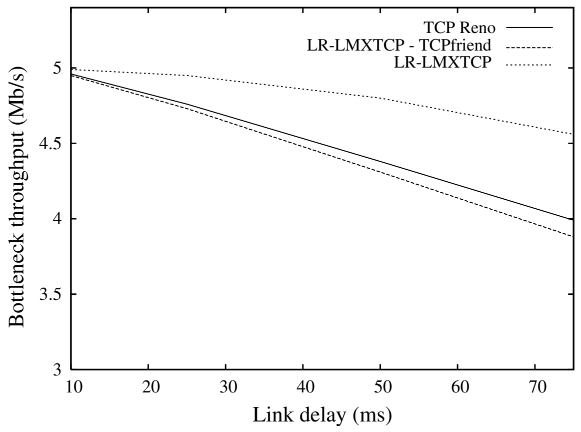


Fig. 9. Bottleneck link utilization as a function of the link delay parameter. Queue size is fixed to 50 packets.

accurate feedback about packet arrival, each connection indeed triggers the retransmissions faster. We conclude that either a finely-tuned congestion control or an accurate loss monitoring is required to face large error rates. In other words, the combination of a coarse *cwnd* adjustment mechanism with cumulative acknowledgments results in a lack of aggressiveness of lossy connections compared to the connections that are not subject to losses.

To complete these results, Figs. 9 and 10 further analyzes the behavior of the TCP-like explicit congestion control algorithms. When the connection bandwidth-delay product increases, Fig. 9 shows that LR-LMXTCP preserves higher utilization of the bottleneck link than TCP or LR-LMXTCP_TCPfriend. This is not surprising since (i) connection backoff is more frequent with TCP than with LR-LMXTCP, and (ii) a backoff penalizes the bottleneck utilization in presence of a large bandwidth-delay product that makes the queues too small to absorb the temporary reduction of sending rate associated to the backoff. Besides, Fig. 10 analyzes the temporal fluctuation of the lossless and lossy connection throughputs for different proto-

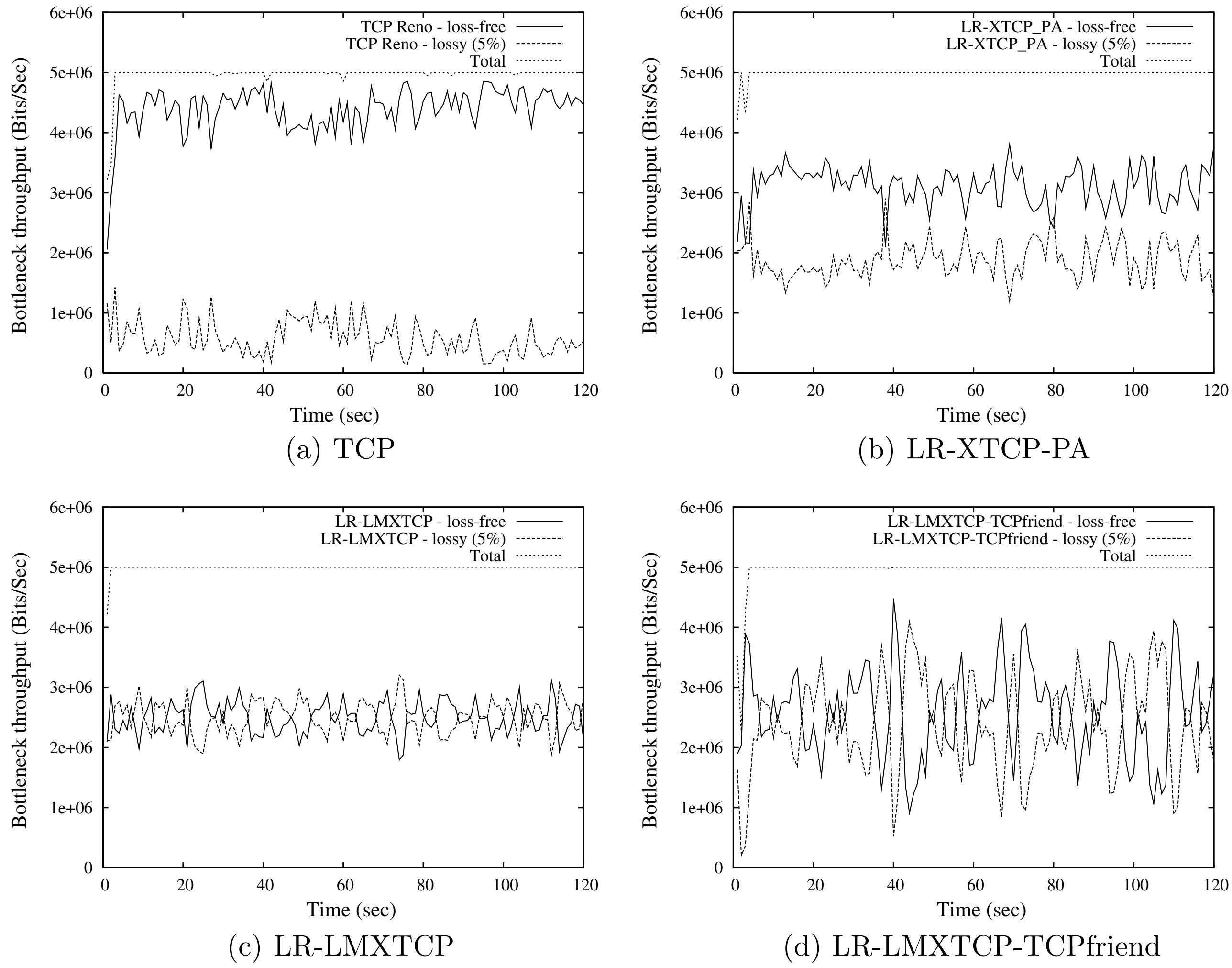


Fig. 10. Sum of loss-free and lossy flow throughputs as a function of time. Losses are randomly distributed with a loss rate of 5%.

cols, and shows that the TCP friendly throughputs fluctuate more than the non-friendly ones. This is in accordance with what we expect from TCP-like connection backoff.

9. Related works

The first paragraph of this section surveys the numerous research efforts that have been made in order to improve TCP performance over lossy links. Because TCP interprets any kind of losses as a congestion notification, without any precaution, TCP results in flow starvation over lossy links. The problem is well-known, especially in wireless environments. In the past decade, three main approaches have been considered to circumvent that problem [13–16]. The first one consists in increasing link-layer reliability to hide link-related losses from TCP sender [25–27]. The second one splits the end-to-end connection, and terminate the TCP connection at the base station to hide the lossy link from the sender [28,29]. The third approach works end-to-end, and attempts to give the TCP sender the capability to handle appropriately losses that are not related to congestion. Some of those schemes builds on refined TCP acknowledgments to allow the TCP sender to recover from multiple losses, without resorting to a coarse timeout [23,30,31]. Others methods distinguish between congestion-related losses and other forms of losses, either based on explicit loss notification [13], or on end-to-end bandwidth estimation [32]. A last strategy proposed to handle non-congestion-related losses consists in delaying or in freezing the congestion response algorithms to allow the recovery of the losses caused by channel errors [16] or hand-offs [15,33]. In a sense, our work is related to the third approach, since we do not attempt to hide losses to the sender, but we rather give the sender the capability to handle them. However, our work is quite different from the approaches described in the above references, essentially because our goal is not to improve TCP performance, but rather to explore the limitations of the window-based congestion control paradigm in lossy environments.

In parallel to solutions for circumventing the fragility of TCP to non-congestion-related losses, a number of works have proposed to decouple congestion control from the observation or inference of losses, similarly to our work. In [12], the size of the congestion window is adapted based on the explicit rate feedback provided by the bottleneck. The authors analyze the performance improvement resulting from the explicit and fine-granular participation of the network in the congestion control. They do not specifically address the loss-resilience problem, and in consequence, they do not encourage the use of aggressive retransmission mechanisms. In [14], the authors exploit the fact that the packet error-rate on wireless links is proportional to the packet size, and propose to control congestion based on the loss patterns observed for tiny TCP/IP header packets. Doing so, they decouple the congestion control from the wireless losses affecting the large data packets. While discussing the performance of their proposed decoupling strategy, the authors in [14] mention that it is important to be aggressive in retransmitting lost data, as long as their transmission is allowed by the congestion window defined based on the small control packets. In that, they stick to one of the main conclusions of our investigations. We go beyond by understanding and alleviating the limits of conventional loss recovery mechanisms when used in conjunction with explicit congestion control mechanisms. Another work of interest is described in [34], where the authors demonstrate that explicit congestion notification (ECN) is unable to distinguish between congestion and wireless losses. They propose to control the flow only based on ECN bits, i.e., without adjusting the congestion window in response to loss packets. In that, the approach is similar to our proposed XTCP. However, the authors in [34] do not discuss the need and relevance for aggressive loss retransmission mechanisms, which is the central component of our contribution. Hence, to the best of our knowledge, none of the earlier works has provided a detailed investigation and a precise description of retransmission mechanisms dedicated to an explicit window-based congestion control framework.

10. Conclusions

We have considered the design of retransmission strategies dedicated to an explicit window-based congestion control framework in the context of two different receiver feedback mechanisms. With conventional cumulative acknowledgment mechanisms, preservation of the *cwnd* upon reception of a duplicate ACK, partial deflation of the send-out window upon reception of a new ACK, and reset of the recovery timer upon reception of both new and duplicate ACKs appear to significantly contribute to maintain the connection efficiency in presence of losses. An original retransmission timer has then been proposed as a complement to the recovery timer to handle multiple losses of the same data segment. In the presence of richer acknowledgments, we have designed loss monitoring and recovery mechanisms capable to exploit precise knowledge about packet reception, in order to increase retransmission aggressiveness.

The proposed approaches have been validated through simulations, both for the XCP protocol introduced in [1], and for an original eXplicit TCP (XTCP) protocol. XCP is characterized by a smooth and precise adjustment of the congestion window in response to accurate and finely-tuned feedbacks computed by XCP routers, while XTCP relies on binary notification about congestion to coarsely adjust the congestion window. The performance of the proposed loss recovery mechanisms are evaluated in terms of the fairness between the lossless and lossy connections sharing a common bottleneck link. Our simulation results have shown that, when the sender is directly notified about packet arrival at the receiver, both XCP and XTCP achieve full utilization and fair partition of the bottleneck resources between lossy and lossless connections. Whilst being always advantageous, a precise notification of received packets to the sender however becomes almost mandatory when the congestion window is controlled based on coarse binary feedback. We conclude that accurate feedback is required either from the router (to support a finely-tuned congestion control) or from the receivers (to monitor losses accurately) in order to preserve the connection efficiency in presence of high loss rates.

The loss-resilience mechanisms proposed in this paper have been shown to maintain close to optimal link utilization, and fair allocation of bottleneck resources among lossy and lossless connections. The combination of explicit control with dedicated retransmission mechanisms provides thus an interesting solution to establish reliable and controlled window-based connections in a lossy environments.

Interestingly, we have also demonstrated that TCP and XTCP may coexist fairly in a single queue of a router, which enables gradual deployment of the proposed explicit control solution. This might be of particular interest when considering wireless infrastructures. In that case, a loss-resilient control protocol might indeed permit to relax the link-layer loss-resilience mechanisms, leaving higher-transport and/or application-layers of the network protocol stack decide about the relevance of a retransmission, thereby resulting in more efficient usage of resources.

References

[1] D. Katabi, M. Handley, C. Rohrs, Internet congestion control for high bandwidth-delay product environments, ACM SIGCOMM (2002) 89–102.

[2] N. Dukkipati, M. Kobayashi, Z.-S. Rui, N. McKeown, Processor sharing flows in the internet, in: Thirteenth International Workshop on Quality of Service (IWQoS), LNCS 3552, Passau, Germany, 2005, pp. 267–281.

[3] N. Dukkipati, N. McKeown, Why flow-completion time is the right metric for congestion control, ACM SIGCOMM Computer Communication Review 36 (1) (2006) 59–62.

[4] V. Jacobson, Congestion avoidance and control, in: ACM SIGCOMM, Stanford, CA, 1988, pp. 314–329.

[5] R.J. La, V. Anantharam, Window-based congestion control with heterogeneous users, in: IEEE INFOCOM, Anchorage, Alaska, 2001, pp. 1320–1329.

[6] M. Allman, V. Paxson, W. Stevens, TCP congestion control, in: RFC 2581, 1999. <http://www.rfc-editor.org/rfc/rfc2581.txt>.

[7] C. Partridge, T. Shepard, Tcp performance over satellite links, IEEE Network 11 (5).

[8] S. Low, F. Paganini, J. Wang, S. Adlakha, J.C. Doyle, Dynamics of tcp/red and a scalable control, in: IEEE INFOCOM, New York, USA, 2002.

[9] M. Allman, D. Glover, L. Sanchez, Enhancing TCP over satellite channels using standard mechanisms, in: RFC 2488, 1999.

[10] M.G., S. Dawkins, M. Kojo, V. Magret, N. Vaidya, Long thin networks, in: RFC 2757, 2000.

[11] K. Ramakrishnan, S. Floyd, A proposal to add explicit congestion notification to IP, in: RFC 2481, 1999.

[12] A. Karnik, A. Kumar, Performance of TCP congestion control with explicit rate feedback, IEEE/ACM Transactions on Networking 13 (1) (2005) 108–120.

[13] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, A comparison of mechanisms for improving TCP performance over wireless links, IEEE/ACM Transactions on Networking 5 (6) (1997) 756–769.

[14] S. Wang, H. Kung, Use of TCP decoupling in improving TCP performance over wireless networks, ACM Wireless Networks 7 (3) (2001) 221–236.

[15] T. Goff, J. Moronski, D. Phatak, V. Gupta, Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments, in: IEEE INFOCOM, Tel-Aviv, Israel, 2000, pp. 1537–1545.

[16] S. Bhandarkar, N. Sadry, A. Narasimha Reddy, N. Vaidya, TCP-DCR: a novel protocol for tolerating wireless channel errors, IEEE Transactions on Mobile Computing 4 (5) (2005) 517–529.

[17] P. Ranjan, E. Abed, R. La, Nonlinear instabilities in tcp-red, in: IEEE INFOCOM, New York, USA, 2002, pp. 249–258.

[18] F. Baccelli, D. Hong, Interaction of TCP flows as billiards, IEEE/ACM Transactions on Networking 13 (4) (2005) 841–853.

[19] S. Floyd, T. Henderson, The Newreno modification to TCP's fast recovery algorithm, in: RFC 2582, 1999.

[20] J.F. Kurose, K.W. Ross, Computer Networking: A Top–Down Approach Featuring the Internet, Addison Wesley, 2001.
[21] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1 (4) (1993) 397–413.
[22] S. Keshav, S. Morgan, SMART retransmission: performance with overload and random losses, INFOCOM'97 3, 1131–1138.
[23] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP selective acknowledgment options, in: RFC 2018, 1996.
[24] C. De Vleeschouwer, P. Frossard, Explicit window-based control in lossy packet networks, Tech. Rep. TR-ITS-2006.016, EPFL – Signal Processing Institute. Available from: <http://lts4www.epfl.ch/~frossard/publications/pdfs/tr_xtcp.pdf> (October 2006).
[25] H. Balakrishnan, S. Seshan, R. Katz, Improving reliable transport and handoff performance in cellular wireless networks, ACM Wireless Networks 1 (4) (1995) 469–481.
[26] E. Amir, H. Balakrishnan, S. Seshan, R. Katz, Efficient TCP over networks with wireless links, in: Fifth workshop on Hot Topics in Operating Systems, 1995, pp. 35–40.
[27] C. Parsa, J. Garcia-Luna-Aceves, Improving TCP performance over wireless networks at the link layer, Mobile Networks and Applications 5 (1) (2000) 57–71.
[28] A. Bakre, B. Badrinath, I-TCP: Indirect TCP for mobile hosts, in: Proceedings of ICDCS, 1995, pp. 136–143.
[29] K. Brown, S. Singh, M-TCP: TCP for mobile cellular networks, IEEE/ACM SIGCOMM Computer Communication Review 27 (5) (1997) 19–43.
[30] K. Fall, S. Floyd, Simulation-based comparisons of Tahoe Reno and SACK TCP, SIGCOMM Computer Communication Review 26 (3) (1996) 5–21.
[31] D. Lin, H. Kung, TCP fast recovery strategies: analysis and improvements, in: INFOCOM, vol. 1, 1998, pp. 263–271.
[32] C. Casetti, M. Gerla, S. Mascolo, M. Sanadidi, R. Wang, TCP Westwood: end-to-end bandwidth estimation for enhanced transport over wireless links, Wireless Networks 8 (5) (2002) 467–479.
[33] W. Lioa, C.-J. Kao, C.-H. Chien, Improving TCP performance in mobile networks, IEEE Transactions on Communications 53 (4) (2005) 569–571.
[34] S. Biaz, X. Wang, Red for improving tcp over wireless networks, in: International Conference on Wireless Networks, Las Vegas, Nevada, USA, 2004, pp. 628–636.

Christophe De Vleeschouwer was born in Namur, Belgium, in 1972. He received the Electrical Engineering degree and the Ph.D. degree from the Université catholique de Louvain (UCL) Louvain-la-Neuve, Belgium in 1995 and 1999 respectively. He was a research engineer with the IMEC Multimedia Information Compression Systems group (1999–2000). He has been a visiting Research Fellow at UC Berkeley (2001–2002), and a post-doctoral researcher at EPFL (2004). He is now a Belgian NSF research associate at UCL in the communication and remote sensing laboratory (TELE). His main interests concern video and image processing for communication and networking applications, including content management and security issues. He is also enthusiastic about non-linear signal expansion techniques, and their use for signal analysis and signal interpretation.

Pascal Frossard is an Assistant Professor at the Signal Processing Laboratory of EPFL (Lausanne, Switzerland), since 2003. He got his M.Sc. and Ph.D., both from EPFL, in 1997 and 2000, respectively. Between 2001 and 2003, he has been with the IBM TJ Watson Research Center (Yorktown Heights, NY, USA). His current research interests include image representation and coding, nonlinear representations, visual information analysis, joint source and channel coding, multimedia communications, and multimedia content distribution.